

# Word Embeddings

Fernando Schiaffino  
schiaffinofernando@gmail.com

Clase 2  
Sábado 07/06/2025

# Word Embeddings

¿Qué son?

- 'Incrustaciones' de palabras
- Representación numérica de los términos de una secuencia en un espacio vectorial
- Intentan capturar información semántica y sintáctica de las palabras
- Palabras cercanas en el espacio tienen un significado / uso similar
- Palabras alejadas en el espacio tienen un significado / uso diferente

# Algunas consideraciones previas

Antes de meternos de lleno en los Word Embeddings revisemos dos nociones.

- Tokenización
- Vectorización

# Tokenización

- Proceso que consiste en dividir una secuencia en unidades mínimas.
- *En general*, podemos establecer una línea entre la idea de token y la idea de palabra.
- Aunque, las arquitecturas más modernas, como veremos, se alejan esta idea.
- Tokenizar:
  - Input: **'Me encantó la película.'**
  - Tokens: [ **'Me', 'encantó', 'la', 'pelicula', ' . '** ]

# Vectorización

Proceso que permite representar un texto con valores numéricos. La idea subyacente a un modelo de embeddings es justamente representar en ese valor aspectos del significado y el uso de un token o una palabra. Para tal efecto, se pueden utilizar técnicas más o menos complejas, con resultados más o menos convincentes.

## Entrada:

*"Me encantó la película, la actuación fue brillante."*

## Lo que la computadora entiende:

Vector numérico:  $[0,2, 0,4, 0,7, 0,1, \dots]$

# Vectorización

- Existen diferentes **técnicas de vectorización**, cuyos vectores resultantes variarán según el método utilizado.
- Cada técnica produce vectores con características y rangos de valores únicos.
- Algunas técnicas producen valores binarios (0 o 1), mientras que otras producen valores continuos entre 0 y 1.

# Técnicas de Vectorización

- Representaciones Básicas
  - One Hot Encoding
  - Bag of words
  - TF-IDF
- Representaciones Avanzadas
  - Word2Vec
  - GloVe
  - FastText
- Representaciones Contextuales
  - BERT
  - Encoder/Decoder
  - LLMs

# One-Hot Encoding

- Representación binaria
- Indica Presencia/Ausencia de cada palabra
- Se genera un vector de dimensión igual al número de palabras en el vocabulario
- Podríamos inferir que cada vector es independiente del resto, o que palabra está representada en su propia dimensión

	el	bar	esta	muy	bueno
el	1	0	0	0	0
bar	0	1	0	0	0
esta	0	0	1	0	0
muy	0	0	0	1	0
bueno	0	0	0	0	1

# Bag Of Words

- Frecuencia de palabras
- Ignora la posición de las palabras
- Considera que todas las palabras son 'independientes' entre sí
- Palabras más frecuentes no necesariamente aportan significado

Documento	el	bar	no	está	muy	bueno	restaurant
El bar no está muy bueno	1	1	1	1	1	1	0
El restaurant está muy bueno	1	0	0	1	1	1	1
<i>Vocabulario</i>	2	1	1	2	2	1	1

# TF-IDF

- Parte de la idea de BOW
- Ajusta la importancia de palabras comunes en el corpus
- Reduce el peso de las palabras muy frecuentes (como stopwords), a la vez que aumenta la importancia de las menos frecuentes (distintivas para un documento).

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \cdot \log\left(\frac{N}{\text{df}(t)}\right)$$

donde:

- $\text{tf}(t, d)$ : Frecuencia de término  $t$  en el documento  $d$
- $N$ : Número total de documentos
- $\text{df}(t)$ : Número de documentos que contienen el término  $t$

# TF-IDF

Ahora consideremos el siguiente escenario:

Documento	Texto crudo	Texto normalizado
Doc1	el bar está muy bueno	bar está bueno
Doc2	el restaurant no está muy bueno	restaurant no está bueno

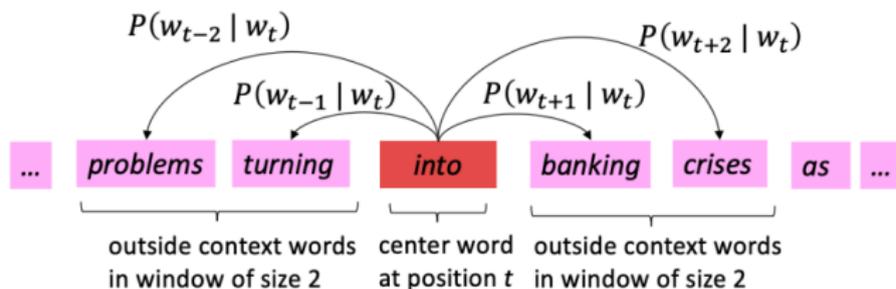
Al aplicar tf-idf obtenemos:

	bar	bueno	está	no	restaurant
<b>Doc1</b>	0.704909	0.501549	0.501549	0.000000	0.000000
<b>Doc2</b>	0.000000	0.409937	0.409937	0.576152	0.576152

- No tiene en cuenta la semántica
- No entiende la negación

# Word2Vec

- Familia de modelos introducida por Mikolov (2013)
- Opera siguiendo la hipótesis distribucional: Palabras con significados similares tienden a aparecer en contextos similares
- “You shall know a word by the company it keeps.” Widdowson (2007)



# Word2Vec

- Este tipo de modelos hacen uso de redes neuronales para aprender las **probabilidades** de encontrar combinaciones de palabras para un contexto dado
- Por cada palabra, el modelo estima la probabilidad de encontrar cada una del resto de palabras del vocabulario en su contexto
- Representan las palabras en un vector denso de 50, 100 o 300 dimensiones
- Este vector no solo representa similitud entre palabras sino que 'captura' el significado de las palabras.

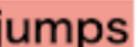
# Word2Vec

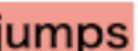
La idea fundamental de Word2Vec es representar cada palabra con dos vectores diferentes:

- Palabra usada como “entrada” (Skip-Gram).
- Palabra en “contexto” (CBOW).

 : Center Word  
 : Context Word

c=0    The cute  jumps over the lazy dog.

c=1    The    over the lazy dog.

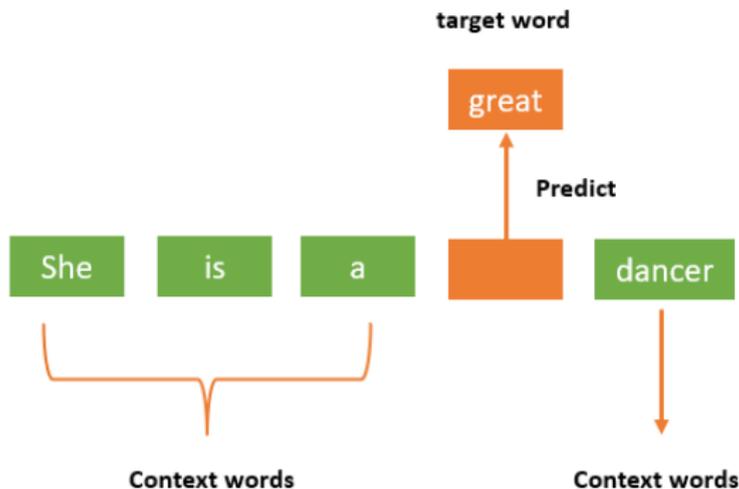
c=2         the lazy dog.

# CBOW (Continuous Bag-of-Words)

**Objetivo:** Predecir la palabra central a partir de su contexto (ventana de palabras vecinas).

**Entrada:** Palabras del contexto (*ventana de contexto*).

**Salida:** Palabra objetivo que se encuentra en el centro del contexto.



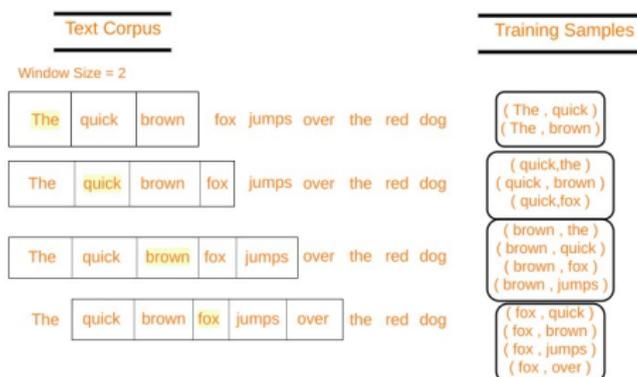
# Skip-Gram (Modelo Word2Vec)

**Objetivo:** Predecir las palabras del contexto a partir de una palabra central.

**Entrada:** Palabra objetivo (central).

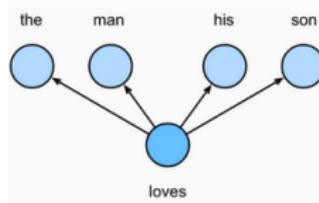
**Salida:** Palabras del entorno cercano (ventana de contexto).

- Captura representaciones más ricas para palabras raras o poco frecuentes.
- Entrenamiento más costoso: genera múltiples pares palabra-contexto por cada palabra central.



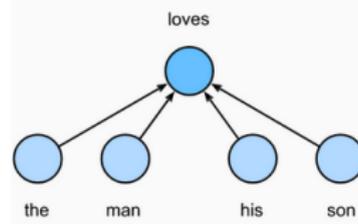
## W2V

## Skipgram



Skipgram necesita menor cantidad de datos y se ha encontrado que representa bien las palabras raras.

## Continuous Bag of Words (CBOW)



CBOW es más rápido y tiene una mejor representación para palabras más frecuentes.

# Entrenamiento

- Supongamos que tomamos todo Wikipedia
- Con esos textos generamos nuestros datasets (tomando pares de palabras en una ventana de contexto deslizante)

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

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

- Además, por cada palabra agregamos una serie de ejemplos negativos tomando palabras random

Pick randomly from vocabulary  
(random sampling)

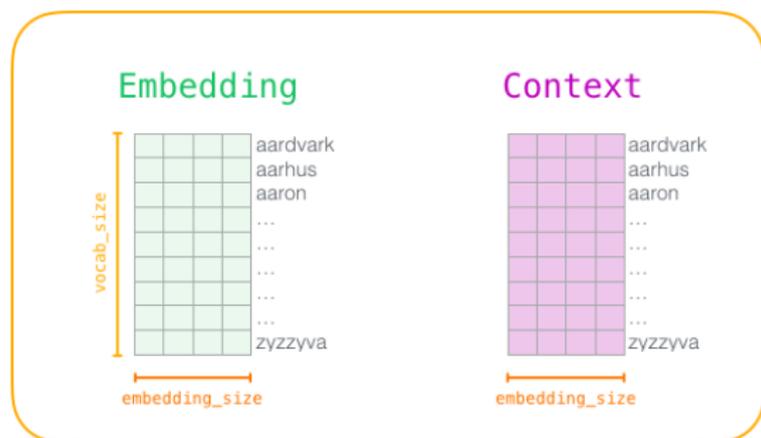
input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Word	Count	Probability
aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		



# Entrenamiento

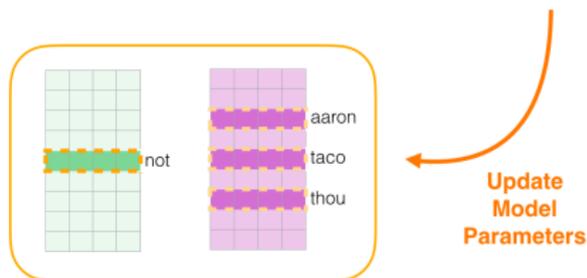
- Recordemos que este modelo representa cada palabra como dos matrices, una cuando la palabra esta siendo usada como central y otra cuando se usa como contexto.



*embedding\_size* representa la dimensión que quiero que tenga mi embedding, *vocab\_size* es el largo del vocabulario total

# Entrenamiento

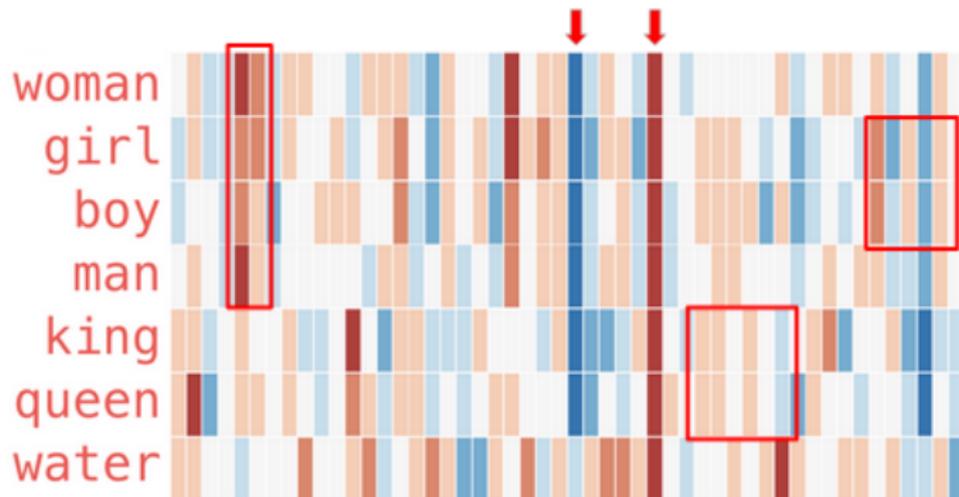
input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



*La red toma los vectores de la palabra central y de las contextuales y realiza operaciones para intentar reducir el error al mínimo*

# Word2Vec

La idea de este tipo de entrenamientos es quedarnos con esos vectores que, luego del entrenamiento, han condensado en sus componentes algunos aspectos relevantes de cada palabra y su distribución.



# Word2Vec

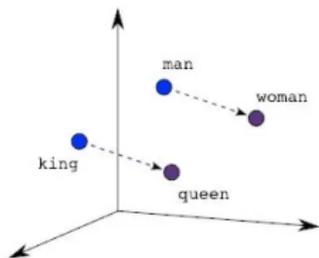
En este sentido los investigadores llegaron a dos resultados:

- **Esperable:** Palabras similares tienen embeddings similares
- **Inesperado:** Estos embeddings eran capaces de capturar información semántica

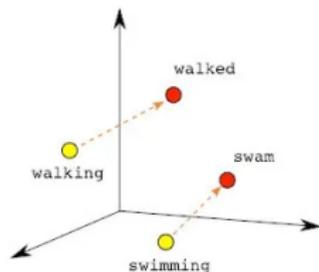
$\text{king} - \text{man} + \text{woman} \approx \text{queen}$



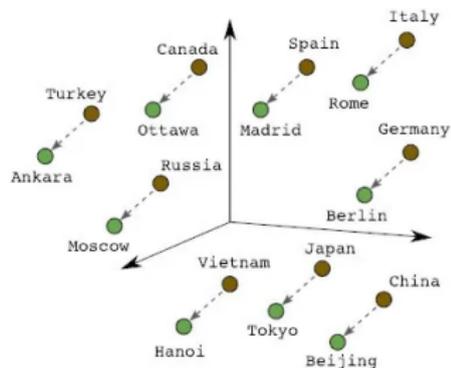
**Figura:** Así, se tomó el vector de la palabra 'Rey', se le restó el vector de la palabra 'Hombre', se le sumó el vector de la palabra 'Mujer' y se obtuvo un vector parecido al de la palabra 'reina'



Male-Female



Verb Tense



Country-Capital

Figura: Además, vemos como estas representaciones permiten establecer relaciones tareas de analogía

# Word2Vec - Visualización

- Veamos ahora cómo se ve un embedding: TensorFlow Embedding Projector

# W2V - Limitaciones

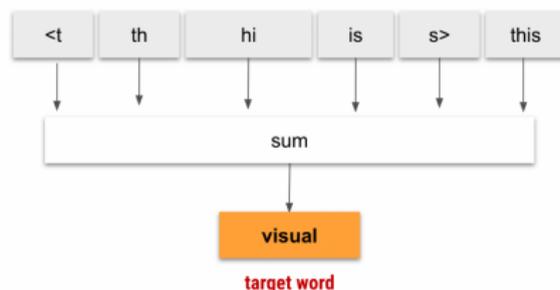
- Si durante el entrenamiento no se ha encontrado un término, W2V no puede crear un vector para él y, en su lugar, asignará un vector aleatorio, lo cual no es óptimo.
- No tiene representaciones compartidas a nivel de subpalabras
- Difícil de escalar a nuevos idiomas

# FastText

- Propuesto por Bojanowski *et al.* (2016)
- A diferencia de W2V, incorpora información de subpalabras
- Captura información de la estructura morfológica
- Cada palabra es representada como un conjunto de n-gramas a nivel del carácter

# FastText: Subwords

## SkipGram + Subword (SkipGramSI)



This is a visual comparison

- Un **n-gram de caracteres** es un conjunto de caracteres co-ocurrentes dentro de una ventana.
- Una **bolsa de n-gramas** representa una palabra como la suma de sus n-gramas.
- Asume implícitamente que cada n-gram es igualmente importante independientemente del contexto, pero, en realidad, ese no es el caso porque no todos los n-gramas son un morfema.

## Recapitulemos:

- Con lo que vimos hasta acá sabemos que existen al menos dos objetivos al entrenar un modelo de embeddings
  - Predecir una palabra dado su contexto
  - Predecir un contexto dado una palabra central
- Tanto W2V como FastText generan Embeddings Estáticos, es decir, más allá de dónde aparezca la palabra, el vector será el mismo
- Ambos utilizan una arquitectura similar que contiene una capa de entrada, una capa intermedia y una capa de salida.
- Entrenan en un gran corpus, ajustando pesos para que las palabras con contextos similares tengan vectores similares.

# Embeddings Contextuales

- ELMo: Embeddings contextuales usando LSTM bidireccional
- Procesan una secuencia de derecha a izquierda y de izquierda a derecha
- Capaz de extraer un embedding para cada palabra dependiendo de la posición en la secuencia
- Transformer y Attention: Permitieron procesar secuencias en paralelo y atender a palabras alejadas sin tener que pasar secuencialmente por cada token

# Modelos basados en Transformers

- Generan embeddings contextuales.
- La misma palabra puede tener diferentes representaciones según el contexto.
- Tienen una comprensión mayor del concepto de ambigüedad y polisemia, ya que generan representaciones en función del contexto.
- Son más potentes, aunque se alejan del concepto tradicional de embeddings fijos.
- Modelos como **BERT** generan vectores a nivel del token, de la oración o del segmento y de la **posición**.

## BERT Embeddings

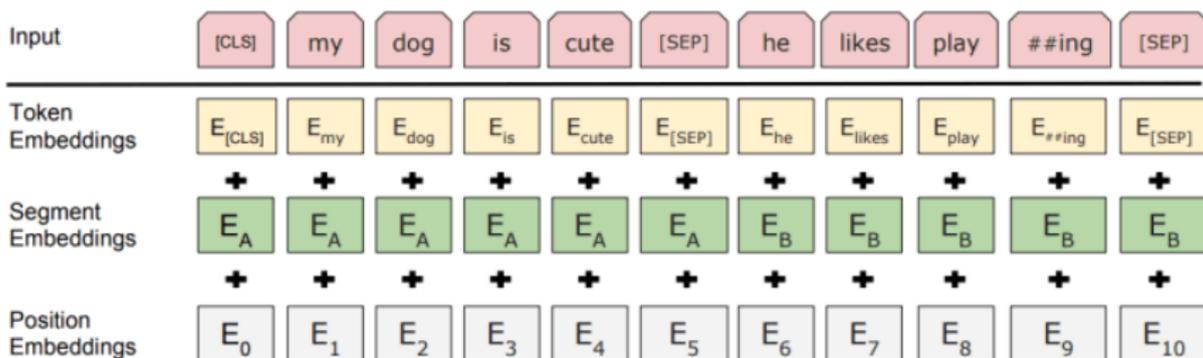


Figura: Aquí podemos ver como esta serie de arquitecturas generan embeddings para diferentes niveles de representación

# Contextual vs. Estáticos

## Word2Vec / FastText (Estáticos)

- Una palabra = un vector, fijo y sin variación por contexto.
- Arquitectura simple, *feed-forward* (Skip-gram / CBOW).
- Buen rendimiento en tareas básicas, muy eficientes.
- Limitación: no distinguen significados múltiples de la misma palabra.

## Embeddings en Transformers (Contextuales)

- Cada palabra (o subpalabra) obtiene un vector distinto según el contexto.
- Basados en **self-attention**, permiten paralelización.
- Mayor capacidad de capturar matices semánticos.
- *Ejemplo*: BERT, GPT, etc. con potencial para tareas complejas.

# Arquitectura y Entrenamiento

- **Word2Vec / FastText:**
  - Modelo **feed-forward** de 2 capas.
  - Objetivo de entrenamiento: predecir el contexto (*Skip-gram*) o la palabra central (*CBOW*).
  - *Subpalabras en FastText*, pero siguen siendo embeddings estáticos.
- **Transformers:**
  - **Autoatención** (*self-attention*) en cada capa, eliminando la recurrencia.
  - Objetivos de entrenamiento diferentes:
    - BERT: *Masked Language Modeling* (MLM).
    - GPT: *Causal Language Modeling* (predicción izquierda a derecha).
  - Embeddings **posicionales** para codificar orden de tokens.

# Ejemplo de Polisemia: *Banco*

## Ilustración de embeddings estáticos vs. contextuales:

### Oraciones:

- “Me senté en el **banco** de la plaza.”
- “Fui al **banco** a sacar dinero.”

### Comparación:

- **Word2Vec:**
  - Mismo vector para “banco” en ambos casos.
- **Transformers (ej. BERT):**
  - **Embeddings diferentes** para “banco” (mueble vs. entidad financiera).
  - Captura el **contexto** alrededor.

*Resultado:* Los modelos contextuales permiten desambiguar sentidos que un embedding estático no distingue.

# Conclusiones y Aplicaciones

- **Word2Vec / FastText:**
  - Fáciles de entrenar, muy rápidos y buenos para tareas simples o recursos limitados.
  - Representaciones **estáticas** no diferencian polisemia.
- **Transformers (LLMs):**
  - Modelos **profundos y contextuales**, apropiados para problemas más complejos (QA, NER, semántica avanzada).
  - Requieren más **recursos de cómputo** y mayor complejidad de implementación.

# Bibliografía I

- Bojanowski, P., Grave, E., Joulin, A., y Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.
- Mikolov, T. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 3781.
- Widdowson, H. (2007). Jr firth, 1957, papers in linguistics 1934–51. *International Journal of Applied Linguistics*, 17(3):402–413.