

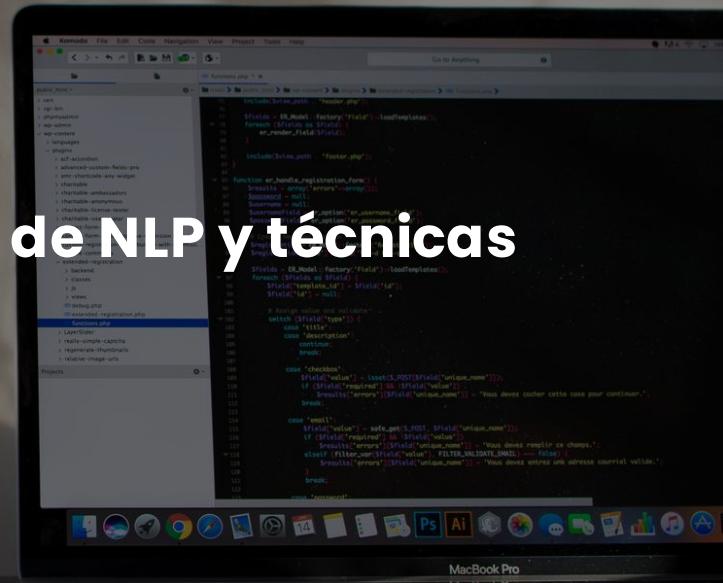
FASTBOOK 04

# Arquitecturas avanzadas de NLP y técnicas de optimización

Introducción al Deep Learning



# Arquitecturas avanzadas de NLP y técnicas de optimización



El presente fastbook se centrará en adquirir los siguientes conocimientos sobre el procesamiento del lenguaje natural (NLP).

- Qué es el procesamiento de lenguaje natural y cuáles son sus principales casos de uso.
- Las principales técnicas para limpiar textos antes de servir como entrada a los modelos.
- Cómo podemos representar las palabras/textos para ser comprendidos por los modelos.
- Los principales modelos de deep learning para crear representaciones de palabras.
- Qué son los modelos secuenciales y cómo mejoran respecto a modelos previos.
- La arquitectura *transformer* y su irrupción como en el campo del NLP.
- Qué es el modelado de lenguaje.
- Cómo y para qué transferir el conocimiento de un modelo a otro.
- Los modelos preentrenados y su mejora respecto a modelos previos.

Y en cuanto a técnicas de optimización de entrenamientos, vamos a aprender:

- La diferencia entre parámetro e hiperparámetro.
- Cuáles son los hiperparámetros principales de un modelo de DL.
- Los métodos para optimizar el proceso de entrenamiento de un modelo.

*Autor: Alejandro Gouloumis*

- Procesamiento de lenguaje natural (NLP)
- Modelos secuenciales (RNNs y LSTMs)
- Transformers
- Aprendizaje por transferencia (transfer learning)
- Modelos preentrenados (pretrained models)
- Técnicas de optimización de entrenamientos
- Conclusión
- Referencias

# Procesamiento de lenguaje natural (NLP)



## Introducción al procesamiento de lenguaje natural (NLP)

El procesamiento de lenguaje natural (NLP, por sus siglas en inglés) es una disciplina que combina los campos de la inteligencia artificial y la lingüística.

Su objetivo es capacitar a las máquinas para entender, interpretar y generar lenguaje humano.

Algunas de las tareas que podemos resolver son la traducción de textos entre idiomas, responder preguntas, resumir grandes textos o reconocer la emoción detrás de las palabras.

Este campo se ha vuelto **un componente esencial en nuestra vida cotidiana**. En la actualidad vemos chatbots que nos hacen creer que estamos hablando con seres humanos, asistentes como Alexa o Siri que resuelven nuestras dudas o herramientas que transforman textos en fotos hiperrealistas.

Aunque la tecnología de NLP está avanzando rápidamente, aún enfrenta desafíos significativos, como sesgos, incoherencias y errores inesperados en su funcionamiento. No obstante, **estas dificultades representan oportunidades que nos permiten aplicar NLP de maneras cada vez más cruciales para nuestra sociedad.** A continuación, vamos a presentar algunos casos de usos del NLP:

### **El análisis de sentimientos**

Se utiliza para analizar opiniones y sentimientos en redes sociales o reseñas de productos, lo que permite entender mejor la percepción pública y mejorar sus productos o servicios.

### **Detección de discursos de odio y moderación de contenido**

El NLP permite identificar y moderar automáticamente contenidos ofensivos o dañinos, contribuyendo a crear entornos digitales más seguros.

### **La traducción automática**

Servicios como Google Translate aplican NLP para traducir textos de un idioma a otro, facilitando la comunicación y el acceso a la información en diferentes idiomas.

## **La clasificación de texto y filtro de spam**

El NLP también permite clasificar automáticamente textos en aquellas categorías relevantes y filtrar correos electrónicos no deseados.

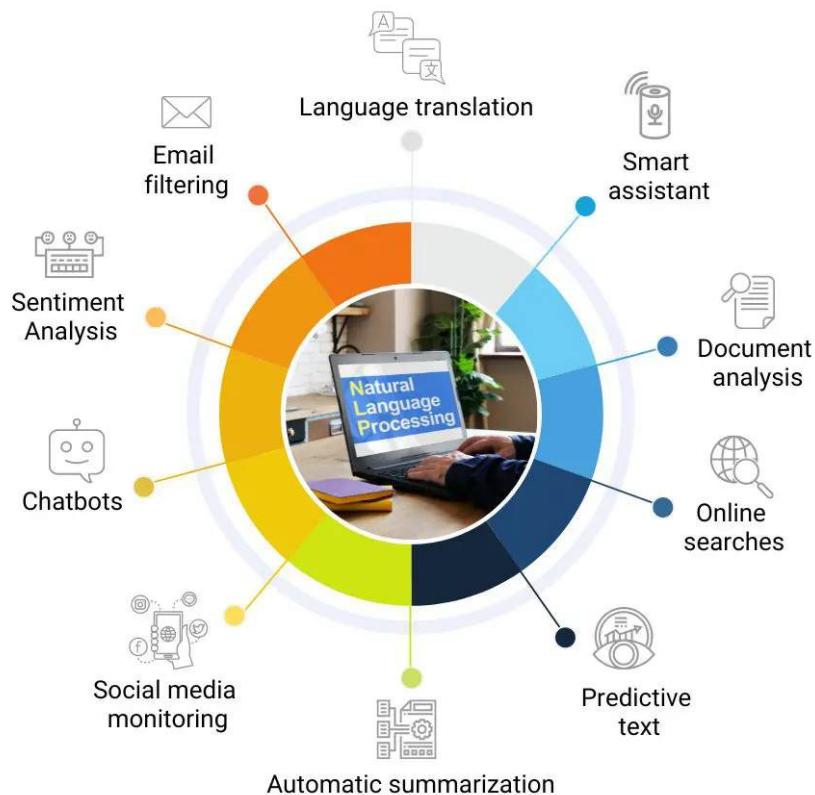
## **Asistentes virtuales y chatbots**

Tal y como se ha comentado, las herramientas como Siri y Alexa utilizan NLP para entender las preguntas de los usuarios y resolver sus dudas. Los servicios de atención al cliente utilizan cada vez más los chatbots para interactuar de manera más rápida con los clientes.

## **El reconocimiento de voz y transcripción**

Gracias al NLP, la conversión de voz a texto y viceversa permite a las personas interactuar con los dispositivos y servicios mediante comandos de voz, además de transcribir automáticamente las conversaciones.

# Applications of Natural Language Processing



Fuente: <https://datasciencedojo.com/blog/natural-language-processing-applications/>.

## Preprocesamiento de los datos

Antes de utilizar un modelo de NLP es esencial **aplicar una serie de procesamientos en los textos para optimizar el desempeño del modelo** o para transformar las palabras y los caracteres en un formato interpretable para el modelo. Estas serían algunas de las transformaciones, ¡anota!

- **Limpieza de texto:** consiste en la eliminación de ruido en los datos, como etiquetas HTML, caracteres especiales, signos de puntuación, etc.
- **Stemming:** proceso en el que se eliminan los sufijos de las palabras para llegar a una forma base o raíz, que no necesariamente tiene que ser una palabra válida en el idioma. Así agrupamos variantes de una palabra que pueden analizarse como un elemento, simplificando el texto. Sin embargo, el stemming puede llevar a errores, como reducir palabras no relacionadas a la misma raíz, por ejemplo: las palabras 'correr', 'corriendo', y 'corredor' podrían ser reducidas al stem 'corr'.
- **Lematización:** proceso más sofisticado que el stemming que implica analizar la morfología de las palabras para encontrar su lema o forma canónica, que debe ser una palabra válida según el diccionario del idioma. La lematización tiene en cuenta el contexto y la parte del discurso de la palabra, lo que resulta en una reducción más precisa y significativa. Este proceso es más complejo y computacionalmente costoso que el stemming, pero proporciona resultados más precisos y útiles para tareas que requieren un alto nivel de comprensión lingüística, por ejemplo: 'vi' (del verbo ver) sería lematizado a 'ver', y 'mejores' a 'bueno'.
- **Segmentación de frases:** proceso que consiste en dividir el texto en oraciones individuales. Resulta útil para tareas que necesitan entender o generar texto a nivel de oración.
- **Eliminación de palabras vacías (stop words):** proceso de eliminación de palabras comunes que aparecen en el idioma, pero que ofrecen poco valor al significado del texto para el análisis, por ejemplo, 'y', 'o', 'pero', etc.
- **Tokenización:** proceso que divide el texto en unidades más pequeñas, como palabras o frases. Cada una de estas unidades se llama token. Un ejemplo sería 'El cielo es azul' que podría quedar como 'El', 'cielo', 'es', 'azul'.

- **Vectorización del texto:** proceso que convierte el texto en una forma numérica para que los algoritmos de machine learning puedan procesarlo. Esto incluye técnicas como la bolsa de palabras (BoW), TF-IDF y embeddings de palabras, como Word2Vec o GloVe.

A continuación, estudiaremos este último proceso: **la vectorización del texto**.

## Vectorización de palabras (*word embeddings*)

Los modelos de machine learning interpretan los datos de una forma diferente a los seres humanos. Mientras que nosotros comprendemos sin problemas frases como "el cielo es azul", estos modelos **no pueden procesar información que no sea numérica**. Por lo tanto, necesitamos transformar el texto que deseamos analizar en números. Los dos métodos más básicos para hacerlo incluyen asignar un número a cada palabra (*one-hot encoding*) o calcular la cantidad de veces que aparece cada palabra en distintos segmentos de texto (*bag of words*).

### **One-hot vectors**

---

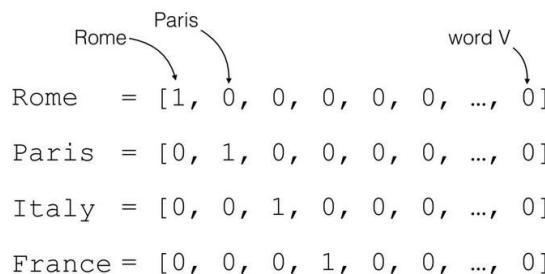
La técnica *one-hot encoding* es la manera más simple de representar palabras como vectores numéricos (*one-hot vectors*).

Para ello primero elegimos un vocabulario de palabras permitidas. Tenemos una tabla que mapea cada palabra del vocabulario a la posición de su *word embedding*. Este *embedding* se puede encontrar usando el índice de la palabra en el vocabulario, de modo que solo un índice tiene un valor distinto de cero. Las palabras que no están incluidas en el vocabulario se les asigna el token 'UNK', del inglés *unknown*, ya que son desconocidas.

Por ejemplo, si tenemos un vocabulario de 4 palabras: 'El', 'cielo', 'es', 'azul', los vectores quedarían de la siguiente manera:

- El: [1,0,0,0,0].
- cielo: [0,1,0,0,0].
- es: [0,0,1,0,0].
- azul: [0,0,0,1,0].
- gato: [0,0,0,0,1].

'Gato' tendría un 1 asignado en la última posición, 'UNK', ya que esta palabra no está contenida en el vocabulario.



Fuente: <https://medium.com/intelligentmachines/word-embedding-and-one-hot-encoding-ad17b4bbell1>.

2

## Bolsa de palabras (*bag of words*)

Los *one-hot vectors* sirven para representar palabras. Para **representar un documento o un fragmento de texto** utilizamos la *bag of words*. Funciona de manera similar.

**CREAMOS UN VOCABULARIO**

**VECTORIZAMOS EL TEXTO**

Se forma una colección exclusiva de palabras extrayendo términos únicos de todos los textos a examinar.

**CREAMOS UN VOCABULARIO**

**VECTORIZAMOS EL TEXTO**

Cada texto se transforma en un vector cuya longitud es la misma que la del vocabulario creado. En este vector, se cuenta la frecuencia de las palabras en el texto.

Por ejemplo, si tenemos las frases 'Me gusta esta película' y 'Esta película es terrible'...



El vocabulario sería el siguiente:

- ❑ Me
- ❑ gusta
- ❑ esta
- ❑ película
- ❑ es
- ❑ terrible

El vocabulario está comprendido de 6 palabras, así que los vectores tendrán una longitud de 6.



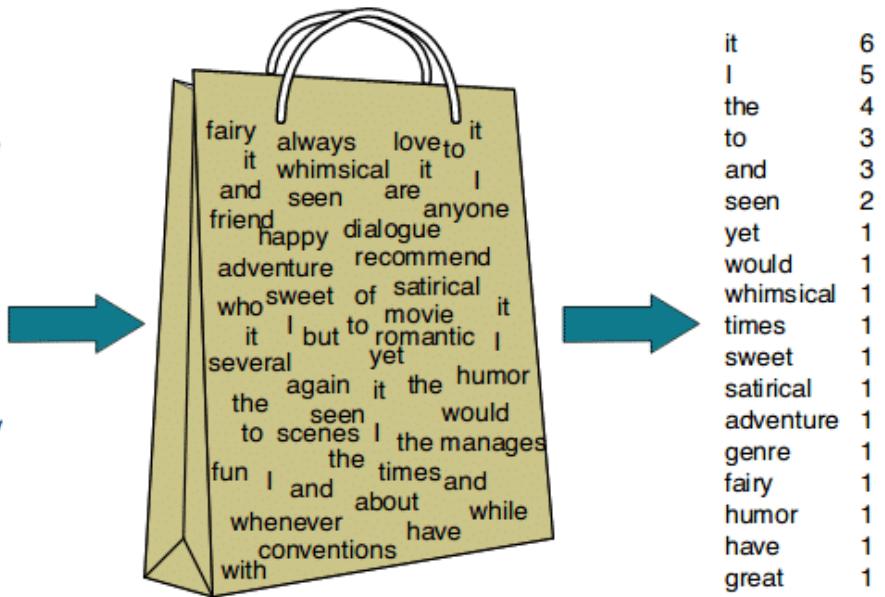
Contamos la frecuencia de las palabras en cada frase:

Término	Me	gusta	esta	película	es	terrible
Fase 1	1	1	1	1	0	0
Fase 2	0	0	1	1	1	1

Por lo tanto, los vectores quedarían:

- 'Me gusta esta película': [1,1,1,1,0,0].
- 'Esta película es terrible': [0,0,1,1,1,1].

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



Fuente: <https://koushik1102.medium.com/nlp-bag-of-words-and-tf-idf-explained-fd1f49dce7c4>.

Esto es solo un ejemplo, en la realidad los vectores tienen dimensiones mucho más grandes, ya que la dimensión del vector es igual al tamaño del vocabulario. Esto hace que sean muy difíciles de escalar con tales tamaños.

3

### TF-IDF

TF-IDF, que significa 'frecuencia de término - frecuencia inversa de documento', pondera la importancia de cada palabra en un documento en relación con una colección de documentos o corpus.

La idea detrás de TF-IDF es **identificar la relevancia de una palabra no solo por su frecuencia en un documento individual (TF), sino también por la rareza de la palabra en todo el corpus (IDF)**.

¿En qué mejora TF-IDF a *bag of words*? *Bag of words* se limita a contabilizar cuántas veces aparecen las palabras del vocabulario en un documento específico. Esto da lugar a que palabras funcionales como artículos, preposiciones y conjunciones, que generalmente tienen un impacto mínimo en el significado global, obtengan una relevancia equiparable a la de palabras con mayor carga semántica, como los adjetivos.

---

**TF-IDF aborda esta limitación de manera efectiva: evita que las palabras con alta frecuencia, pero de menor relevancia semántica, opaquen a aquellas menos frecuentes, pero de mayor importancia.**

La terminología que usaremos será la siguiente:

- t → término (palabra).
- d → documento (conjunto de palabras).
- N → recuento del corpus.
- corpus → conjunto total de documentos.

Ahora, estudiaremos cada una de las partes de TF-IDF.

### Frecuencia de término (TF)

Mide la frecuencia de una palabra en un documento. Cuanto más aparezca una palabra, más importante se considera para ese documento. Sin embargo, si una palabra aparece muchas veces en muchos documentos, será menos relevante. Es por ello por lo que necesitamos la frecuencia inversa de documento (IDF), pero antes debemos conocer la frecuencia de documento (DF).

$$tf(t,d) = \text{frecuencia de } t \text{ en } d / n^o \text{ total de palabras en } d$$

### Frecuencia de documento (DF)

Mide la importancia de los documentos en el corpus completo. Es similar a TF con la única diferencia de que TF cuenta la frecuencia de un término en el documento, mientras que DF es el número de documentos en que una palabra está presente.

$$df(t) = \text{frecuencia de } t \text{ en } N \text{ documentos}$$

### Frecuencia inversa de documento (IDF)

A la inversa de DF, IDF mide la rareza de una palabra en el corpus completo. Este valor será muy pequeño para palabras muy frecuentes como las stop words (porque están presentes en casi todos los documentos).

$$idf(t) = N / df$$

Pero existe un problema: cuando manejamos un corpus extenso, por ejemplo, de 10.000 documentos, el valor de IDF puede incrementarse considerablemente. Para moderar este aumento, aplicamos el logaritmo a IDF.

$$idf(t) = \log(N/df)$$

Durante una búsqueda, si la palabra buscada no forma parte del vocabulario, se omite. No obstante, en situaciones donde se emplea un vocabulario predefinido y ciertas palabras de este no figuran en el documento, el DF sería 0. Para evitar la división por cero, se ajusta el valor incrementando en 1 el denominador.

$$idf(t) = \log(N/(df + 1))$$

Al combinar de forma multiplicativa el TF e IDF, se calcula la puntuación de TF-IDF. Aunque existen múltiples versiones de TF-IDF, nos enfocaremos en esta formulación elemental por el momento:

$$tf-idf(t, d) = tf(t, d) * \log(N/(df + 1))$$

## Semántica distribucional

Existe un gran problema con los métodos basados en frecuencias estudiados anteriormente: estas representaciones **no contienen información sobre las palabras que representan**. Es decir, la palabra 'cielo' sería tan parecida a la palabra 'azul' como a 'gato'. Esto viene a decir que estas representaciones no capturan el significado.

Para comprender cómo las palabras adquieren significado en sus representaciones vectoriales, primero necesitamos establecer una definición práctica de ‘significado’. Para lograr esto, exploraremos el proceso mediante el cual nosotros, como seres humanos, discernimos qué palabras poseen un significado parecido.

Por ejemplo:

- Es demasiado *tzatziki* para mí.
- El *tzatziki* sirve de acompañamiento de muchos platos.
- El *tzatziki* sirve de acompañamiento de muchos platos.
- Mi madre hace muy buen *tzatziki*.

Aunque no conozcas la palabra marcada en rojo, has podido entender su significado gracias a los diferentes contextos en los que se ha utilizado la palabra. Tu cerebro ha buscado otras palabras que pueden usarse en los mismos contextos, y ha llegado a la conclusión de que su significado es similar a esas otras palabras. Esta es la hipótesis distributiva:

---

**Las palabras que aparecen en contextos similares tienen significados similares, o, dicho de otra forma, el significado de una palabra viene dado por las que aparecen frecuentemente cerca de ella.**

---

A veces se puede encontrar formulada como "You shall know a word by the company it keeps" (J. R. Firth 1957: 11).

Esta es **una idea tremadamente útil**: se puede aplicar en la práctica para que los vectores de palabras reflejen su significado. De acuerdo con la hipótesis distribucional, entender el significado y comprender los contextos son fundamentalmente lo mismo. Así, lo único que necesitamos es incorporar información sobre los contextos de las palabras en su representación.

## 5

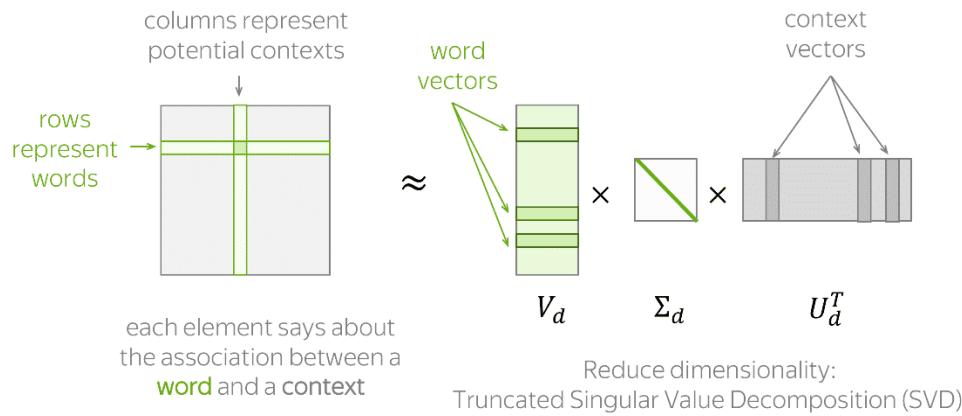
### Métodos basados en el recuento

---

Los métodos basados en el recuento toman esta idea al pie de la letra: poner esta información manualmente basándose en las estadísticas globales del corpus.

El método general consiste en **dos fases principales**:

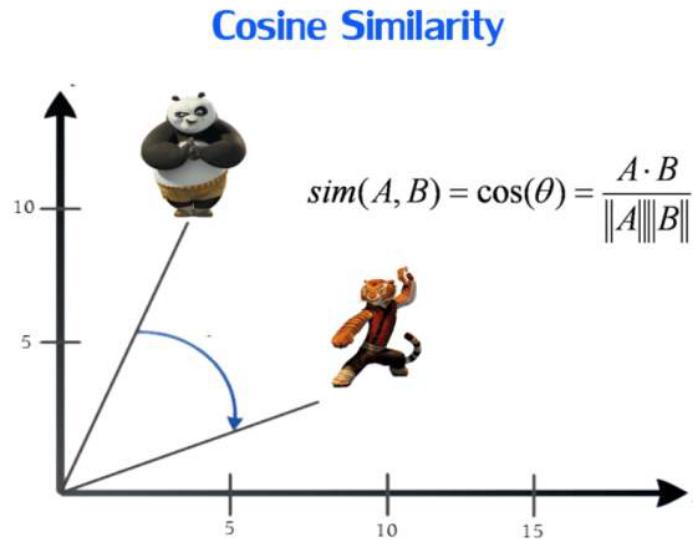
- Crear una matriz de palabras por contexto.
- Reducción de su dimensionalidad.



Fuente: [https://lena-voita.github.io/nlp\\_course/word\\_embeddings.html](https://lena-voita.github.io/nlp_course/word_embeddings.html).

La necesidad de **reducir la dimensionalidad surge por dos razones**: la matriz original suele ser muy grande y, al incluir muchas palabras con contextos limitados, contiene numerosos elementos que no aportan información relevante (como los ceros).

Para evaluar la similitud entre palabras o contextos, es común recurrir al cálculo del producto escalar de vectores normalizados, es decir, la similitud coseno.



<sup>1</sup> Image courtesy techninpink.com

Fuente: <https://datascience103579984.wordpress.com/2020/01/19/feature-engineering-for-nlp-in-python-from-datacamp/4/>.

Para implementar un enfoque basado en el recuento, es necesario definir dos aspectos fundamentales:

- Los **contextos posibles** (o sea, qué implica que una palabra figure en un contexto determinado).
- La **idea de asociación**, esto es, las ecuaciones empleadas para determinar los valores de la matriz.

Los métodos más comunes para realizar estas definiciones son los que presentamos a continuación, ¡apunta!

### **Co-ocurrencias usando una ventana**

Consiste en crear una matriz cuyas filas y columnas representan un único elemento del corpus. Las celdas de la matriz hacen referencia al número de veces que dos elementos aparecen juntos en un contexto.

-	apples	are	green	and	red	sweet	oranges	sour
apples	2	2	1	1	2	1	0	0
are	2	3	1	1	2	1	1	1
green	1	1	2	1	1	0	1	1
and	1	1	1	1	1	0	0	0
red	2	2	1	1	2	1	0	0
sweet	1	1	0	0	1	1	0	0
oranges	0	1	1	0	0	0	1	1
sour	0	1	1	0	0	0	1	1

### Información mutua puntual (PPMI)

En esta técnica los contextos se establecen de la igual forma que se hizo previamente. Sin embargo, el método utilizado para determinar la relación entre una palabra y su contexto es el índice de probabilidad mutua positiva (PPMI).

### Análisis semántico latente (LSA)

Este método examina un conjunto de documentos. A diferencia de los métodos previos donde los contextos se utilizaban únicamente para crear los vectores de palabras y luego se descartaban, en LSA el contexto, o también llamados 'los vectores de documentos', también son de interés.

6

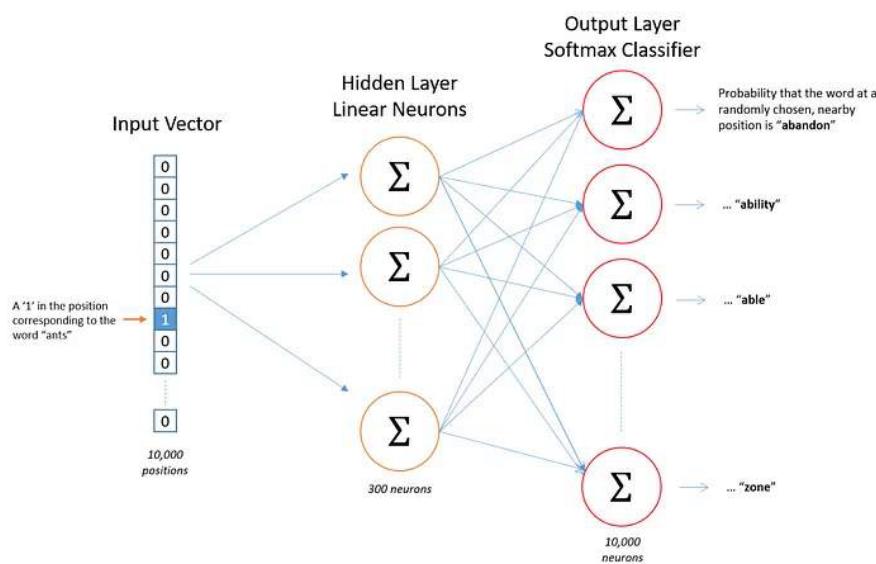
## Word2vec

En 2013 un grupo de investigadores de Google revolucionó el mundo de los *word embeddings* (2013b, a), quienes propusieron los modelos CBOW y *skip-gram*, popularmente conocidos como Word2vec. En pocas palabras, este enfoque utiliza la potencia de una simple red neuronal para generar *word embeddings*. Esto supuso una mejora sustancial en comparación con los métodos basados en frecuencia (*bag of words*, TF-IDF), ya que por fin se tuvo en cuenta el contexto en la representación vectorial de las palabras.

Word2vec consiste en **una red neuronal poco profunda** de dos capas. Su funcionamiento es muy básico:

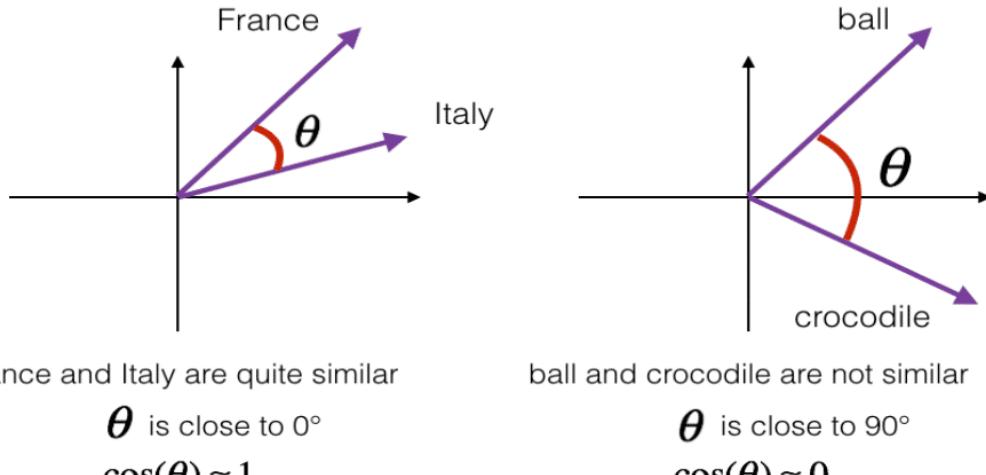
- Toma como entrada un extenso conjunto de textos.
- Recorre el texto con una ventana deslizante, moviendo una palabra cada vez. En cada paso, hay una palabra central y palabras de contexto (otras palabras en esta ventana).
- Para la palabra central, calcula las probabilidades de las palabras de contexto.
- Ajusta los vectores para aumentar estas probabilidades.

Como ocurre con cualquier red neuronal, posee pesos que, durante el entrenamiento, se ajustan para **minimizar una función de pérdida**. No obstante, el uso de Word2Vec no es para la función específica para la que se entrenó. En su lugar, tomamos solo los pesos de la capa oculta, que se usan como nuestras representaciones vectoriales de palabras, y descartamos el resto del modelo.



Fuente: <https://medium.com/@Aj.Cheng/word2vec-3b2cc79d674>.

En este espacio vectorial, **las palabras que frecuentemente coexisten en los mismos contextos están ubicadas próximas entre sí**. Esto se calcula con lo que hablamos en el apartado anterior sobre la similitud coseno.

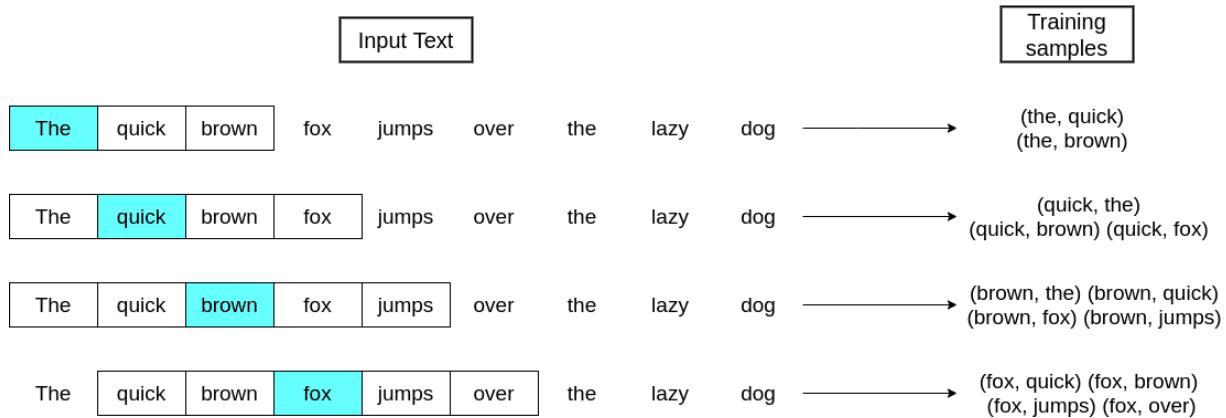


Fuente: <https://medium.com/analytics-vidhya/best-nlp-algorithms-to-get-document-similarity-a5559244b23b>.

El modelo se presenta en **dos variantes**: el modelo de bolsa de palabras continua (CBOW) y el modelo Skip-gram. A nivel algorítmico, ambos modelos comparten similitudes:



**Skip-gram** predice el contexto dada una palabra central. La siguiente imagen ilustra los ejemplos de entrenamiento que generaría con esa frase y un tamaño de ventana = 2.

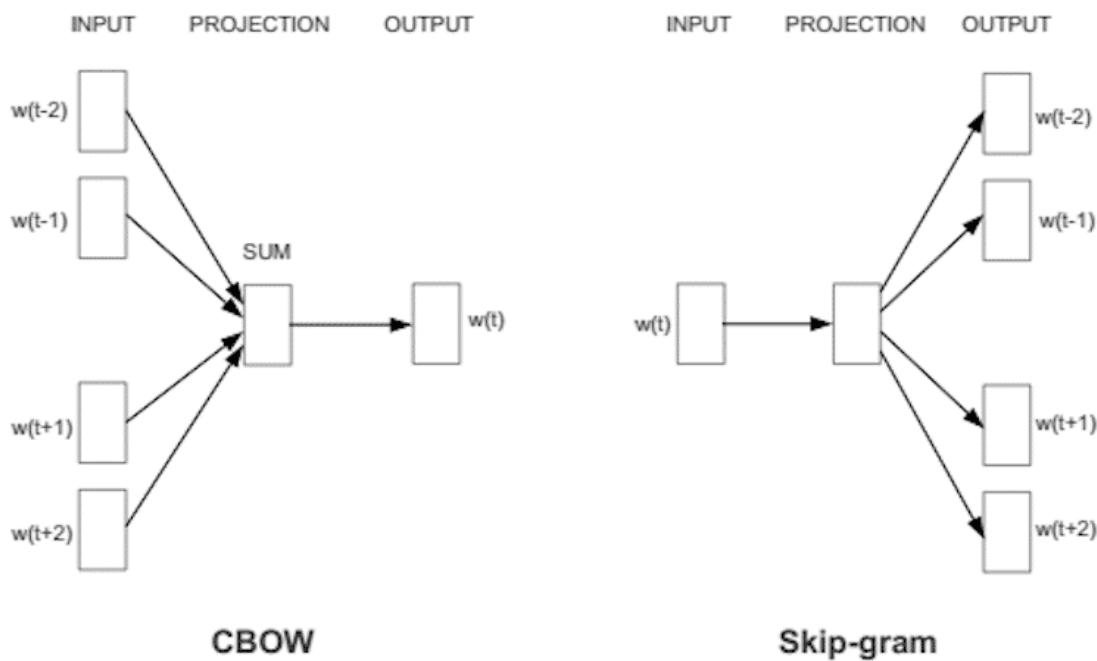


Fuente: <https://neptune.ai/blog/vectorization-techniques-in-nlp-guide>.

---



**Continuous bag-of-words (CBOW)** predice la palabra central sumando los vectores del contexto (inversa de skip-gram).



Fuente: <https://arxiv.org/pdf/1301.3781.pdf>.

---

¿Y cuándo usamos un modelo u otro? De acuerdo con el *paper* original, el modelo skip-gram muestra un buen desempeño con conjuntos de datos de menor tamaño y logra una representación más precisa de palabras menos comunes. Por otro lado, se ha observado que CBOW se entrena más rápidamente en comparación con skip-gram y es más efectivo al representar palabras frecuentes. Así, la decisión entre utilizar skip-gram o CBOW varía según la naturaleza del problema a abordar.



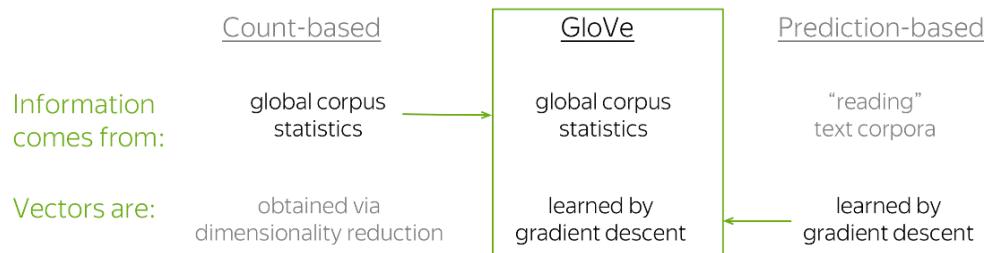
En este [link](#) hay una explicación visual e intuitiva de word2vec.

7

## GloVe

El modelo GloVe (2014), diseñado en la Universidad de Stanford, es una combinación de métodos basados en el recuento y métodos predictivos (Word2vec).

Su nombre significa ‘global vectors’, que refleja su idea: utiliza información global del corpus para aprender los vectores.



Fuente: [https://lena-voita.github.io/nlp\\_course/word\\_embeddings.html](https://lena-voita.github.io/nlp_course/word_embeddings.html).

## ¿Cómo supera GloVe a Word2vec?

Word2vec se centra en el contexto inmediato, dependiendo de una ventana específica para formar los embeddings de las palabras. Esto restringe el aprendizaje de una palabra a las que se encuentran cercanas en el texto, limitando así el aprovechamiento de la información disponible, ya que solo se utiliza un fragmento de todo el conocimiento posible. Por otro lado, GloVe mejora este proceso al integrar tanto información de contexto local como global al construir los vectores de palabras, aprovechando de manera más eficiente las estadísticas disponibles.

## ¿Cuándo se usa GloVe?

GloVe supera a otros modelos en tareas de analogía de palabras, similitud de palabras y reconocimiento de entidades con nombre (NER). Como incorpora estadísticas globales, puede captar la semántica de palabras raras y obtiene buenos resultados incluso con un corpus pequeño.

8

## Cómo evaluar los *embeddings*

Podemos evaluar los *embeddings* de **dos maneras**: de manera intrínseca o extrínseca.

**INTRÍNSECA****EXTRÍNSECA**

Este tipo de evaluación se fija en las propiedades internas de los *embeddings*, por ejemplo, cómo de bien capturan el significado. Es más rápido de evaluar, pero no te dice cuál es mejor en la práctica.

**INTRÍNSECA****EXTRÍNSECA**

Este tipo de evaluación te dice qué *embeddings* son mejores para la tarea que te interesa, por ejemplo, en la clasificación de texto. El inconveniente es que tienes que entrenar el modelo/goritmo para una tarea varias veces: un modelo para cada uno de los *embeddings* que quieras evaluar, para así decidir qué *embeddings* son mejores viendo la calidad de los modelos.

No existe un tipo de evaluación mejor que otro, depende de muchos factores. Sin embargo, **la evaluación extrínseca es más interesante** ya que te dice qué modelo genera los mejores *embeddings* a la hora de la práctica. El inconveniente es que tienes que entrenar varios modelos.

## Relaciones semánticas y sintácticas

Los espacios semánticos buscan generar representaciones de lenguaje natural que reflejen su significado.

Podemos considerar que las representaciones vectoriales de palabras efectivas constituyen un espacio semántico, y nos referimos a una colección de vectores de palabras dentro de un espacio multidimensional como 'espacio semántico'.

En términos prácticos, si varias palabras comparten contextos parecidos, Word2Vec y GloVe deberían generar resultados análogos al introducir estas palabras, y para lograr resultados semejantes, los vectores correspondientes a dichas palabras (creados en la capa oculta) deben ser parecidos. Esto implica que estos modelos se diseñan para formar vectores de palabras afines para términos que aparecen en contextos similares.

Closest to **frog**:

frogs  
toad  
litoria  
leptodactylidae  
rana  
lizard  
eleutherodactylus

litoria



leptodactylidae



rana



eleutherodactylus

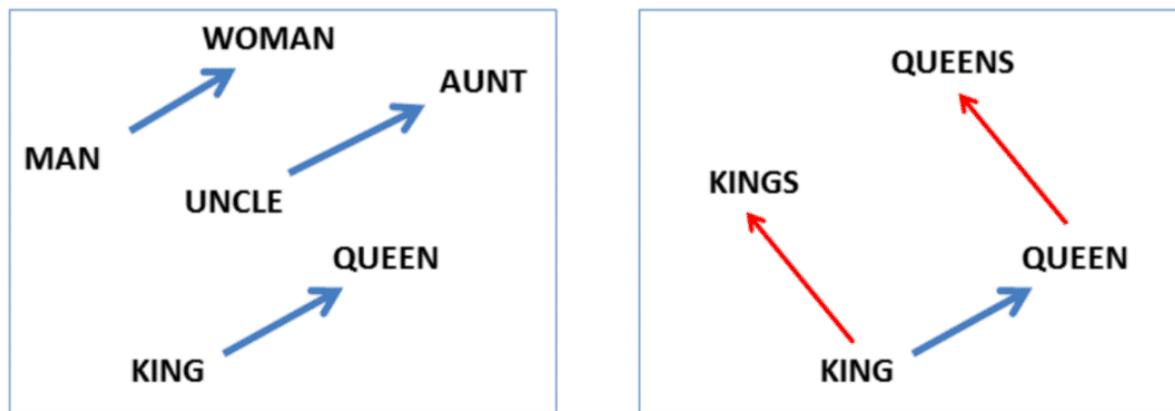


Podemos asumir que las relaciones semánticas y sintácticas son lineares en el espacio vectorial. Por ejemplo, la diferencia entre 'rey' y 'reina' es casi la misma que entre 'hombre' y 'mujer'. Algunos ejemplos:

- $v(\text{rey}) - v(\text{hombre}) + v(\text{mujer}) = v(\text{reina})$
- $v(\text{reyes}) - v(\text{rey}) + v(\text{reina}) = v(\text{reinas})$
- $v(\text{España}) - v(\text{Madrid}) + v(\text{París}) = v(\text{Francia})$
- $v(\text{fuego}) - v(\text{caliente}) + v(\text{frío}) = v(\text{hielo})$

semantic:  $v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$

syntactic:  $v(\text{kings}) - v(\text{king}) + v(\text{queen}) \approx v(\text{queens})$



Fuente: [https://lena-voita.github.io/nlp\\_course/word\\_embeddings.html](https://lena-voita.github.io/nlp_course/word_embeddings.html).

Se ha demostrado que los *word embeddings* contienen sesgos ('Man is to computer programmer as woman is to homemaker?'). Al final los datos de entrada para un modelo de aprendizaje definen su percepción de la realidad. Si un modelo se entrena con estos datos bajo la premisa de que representan lo que es, interpretará estos datos como una norma a seguir, convirtiéndose en una herramienta que perpetúa el *status quo* con todas sus imperfecciones.

Este fenómeno se vuelve particularmente marcado en el caso de los *word embeddings*, ya que estas abstraen las palabras de los contextos originales en los que fueron usadas, dejando como resultado un sesgo estático inherente. Algunos ejemplos tienen que ver con las profesiones más comunes para hombres y mujeres:

- Oficios de mujeres: empleada de hogar, enfermera, recepcionista...
- Oficios de hombres: maestro, filósofo, capitán...

#### **Extreme *she* occupations**

- |                 |                       |                        |
|-----------------|-----------------------|------------------------|
| 1. homemaker    | 2. nurse              | 3. receptionist        |
| 4. librarian    | 5. socialite          | 6. hairdresser         |
| 7. nanny        | 8. bookkeeper         | 9. stylist             |
| 10. housekeeper | 11. interior designer | 12. guidance counselor |

#### **Extreme *he* occupations**

- |                |                   |                |
|----------------|-------------------|----------------|
| 1. maestro     | 2. skipper        | 3. protege     |
| 4. philosopher | 5. captain        | 6. architect   |
| 7. financier   | 8. warrior        | 9. broadcaster |
| 10. magician   | 11. fighter pilot | 12. boss       |

Fuente: <https://www.semanticscholar.org/paper/Man-is-to-Computer-Programmer-as-Woman-is-to-Word-Bolukbasi-Chang/ccf6a69a7f33bcf052aa7def176d3b9de495beb7>.

# Modelos secuenciales (RNNs y LSTMs)



Antes de explicar qué son los modelos secuenciales y por qué se usan para tareas de NLP, hace falta hablar sobre el modelado de lenguaje. Básicamente, esta tarea se encarga de predecir la palabra que viene después. Por ejemplo, si tenemos como contexto:

- El cielo es \_\_\_\_\_

Las posibles respuestas pueden ser 'azul', 'bonito', etc; y formalmente se define como:

- 'Dada una secuencia de palabras, calcular la probabilidad de la siguiente palabra, donde ésta puede ser cualquier palabra del vocabulario'

---

Un sistema que realiza esta tarea se denomina modelo de lenguaje (LM). También se puede pensar en un LM como un sistema que asigna una probabilidad a una pieza de texto.

Estos modelos están presentes en nuestra vida cotidiana, un ejemplo es el teclado predictivo de Google.



Fuente: <https://t21.com.mx/tecnologia-2016-12-19-gboard-teclado-predictivo-google/>.

Algunas de las aproximaciones de los LM son los modelos de n-gramas basados en recuento, modelos neuronales de n-gramas o redes neuronales recurrentes. A continuación, estudiaremos el uso de estas últimas en el NLP.

## Redes neuronales recurrentes (RNNs)

En temas anteriores ya hemos estudiado qué son **las redes neuronales recurrentes** (RNNs). Ahora veremos por qué son útiles para resolver diferentes tareas de NLP.

El problema de las redes neuronales *feedforward* radica en su presuposición de que las longitudes de los vectores de entrada y salida son fijas y conocidas de antemano. Sin embargo, para diversas aplicaciones en el procesamiento del lenguaje natural, como la traducción automática y el reconocimiento de voz, determinar las dimensiones exactas con antelación resulta inviable. Es por ello por lo que se requieren modelos capaces de mapear secuencias de palabras en otras secuencias de palabras (Sutskever, Vinyals, y Le 2014).

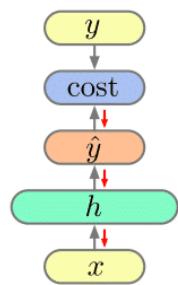
---

Los modelos secuenciales (*seq2seq*) como las RNNs se consideran modelos de lenguaje condicional (CLM) y no LM, ya que estiman la probabilidad condicional  $p(y|x)$  de una secuencia 'y' dada la fuente 'x'.

Los CLM funcionan de manera similar a los LM, pero reciben además información sobre la fuente.

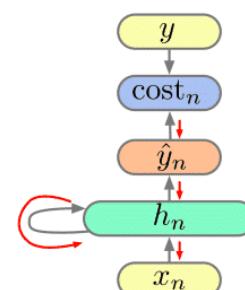
Feed Forward

$$\begin{aligned} h &= g(Vx + c) \\ \hat{y} &= Wh + b \end{aligned}$$



Recurrent Network

$$\begin{aligned} h_n &= g(V[x_n; h_{n-1}] + c) \\ \hat{y}_n &= Wh_n + b \end{aligned}$$



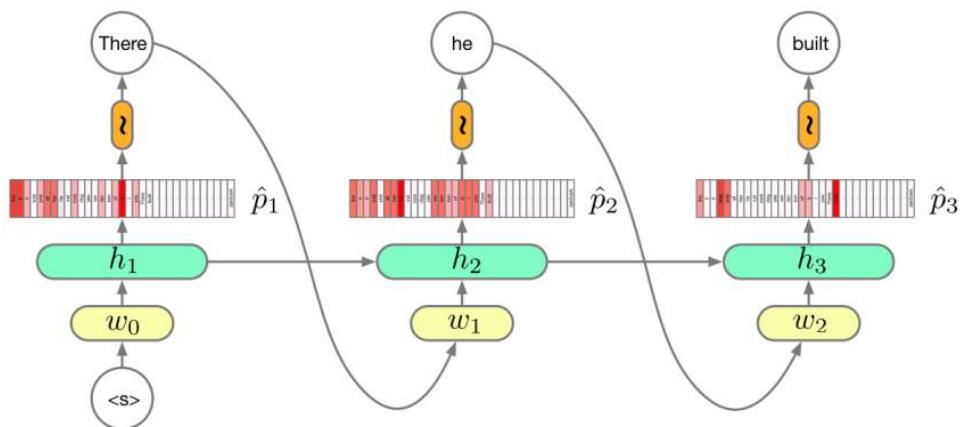
Fuente: Phil Blunsom. Deep Learning for Natural Language Processing. Oxford University and DeepMind. 2017.

Estas son algunas de las **ventajas** de las RNNs:

- Pueden procesar entradas de cualquier longitud.
- El tamaño del modelo no aumenta con el tamaño de la entrada.
- Cada paso del entrenamiento tiene en cuenta la información histórica.
- Los pesos se comparten a lo largo del tiempo.

Las **desventajas** son básicamente tres:

- Son lentos computacionalmente.
- Dificultad para acceder a información de hace mucho tiempo (secuencias largas).
- No puede considerar ninguna entrada futura para el estado actual.



Fuente: Phil Blunsom. Deep Learning for Natural Language Processing. Oxford University and DeepMind. 2017.

Ahora vamos a estudiar los tipos de RNNs y sus usos, ¡anota!

### One to many

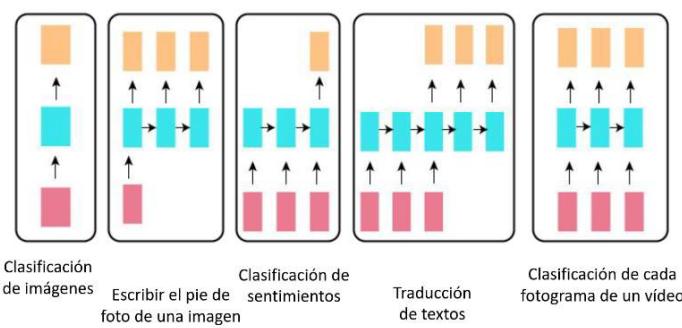
Aquí el dato de entrada tiene un formato estándar y no es una secuencia, pero la salida sí es una secuencia (por ejemplo, el subtitulado de imágenes).

### Many to one

En esta RNN el dato de entrada es una secuencia, pero la salida es un vector de tamaño fijo, no una secuencia (por ejemplo, el análisis de sentimientos).

### Many to many

Aquí el dato de entrada y de salida son secuencias (por ejemplo, la traducción de textos o clasificación de cada fotograma de un vídeo).



Fuente: [https://github.com/somosnlp/nlp-de-cero-a-cien/blob/main/2\\_modelos\\_secuenciales/modelos\\_secuenciales.pdf](https://github.com/somosnlp/nlp-de-cero-a-cien/blob/main/2_modelos_secuenciales/modelos_secuenciales.pdf)

Sin embargo, existe un problema con las RNNs a través de los gradientes.

Recordemos que el gradiente indica el ajuste a realizar en los pesos con respecto a la variación en el error. Este problema puede venir en dos tipos diferentes:

#### **GRADIENTES EXPLOSIVOS O 'EXPLODING GRADIENTS'**

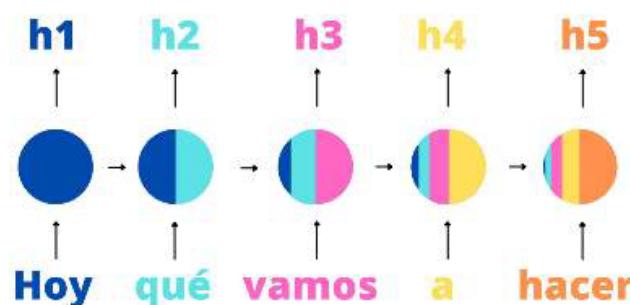
#### **GRADIENTES DESAPARECIDOS O 'VANISHING GRADIENTS'**

Cuando el algoritmo asigna una importancia exageradamente alta a los pesos. En este caso, el problema se puede resolver fácilmente truncando los gradientes (*gradient clipping*).

#### **GRADIENTES EXPLOSIVOS O 'EXPLODING GRADIENTS'**

#### **GRADIENTES DESAPARECIDOS O 'VANISHING GRADIENTS'**

Los valores de los gradientes son demasiados pequeños y el modelo deja de aprender o aprende muy despacio.



Fuente: [https://github.com/somosnlp/nlp-de-cero-a-cien/blob/main/2\\_modelos\\_secuenciales/modelos\\_secuenciales.pdf](https://github.com/somosnlp/nlp-de-cero-a-cien/blob/main/2_modelos_secuenciales/modelos_secuenciales.pdf)

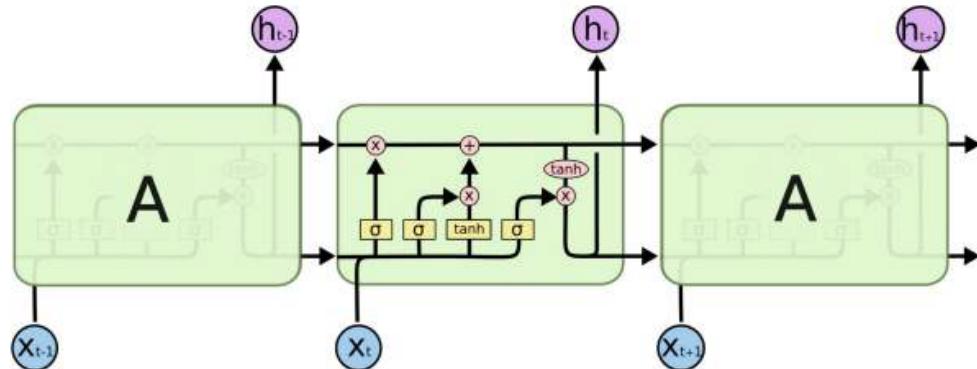
---

**Recordamos que en cada paso la RNN transmitirá parte de lo aprendido al siguiente paso. Pues bien, en la imagen mostrada, vemos en el último paso ‘h5’ como la información de la palabra ‘Hoy’ es muy pequeña, siendo esta palabra muy importante para el significado de la oración.**

Es un ejemplo, ya que esto sucede a menudo cuando las secuencias son relativamente largas. Aquí es donde entra una variante de las RNNs, las *long short-term memory* (LSTMs).

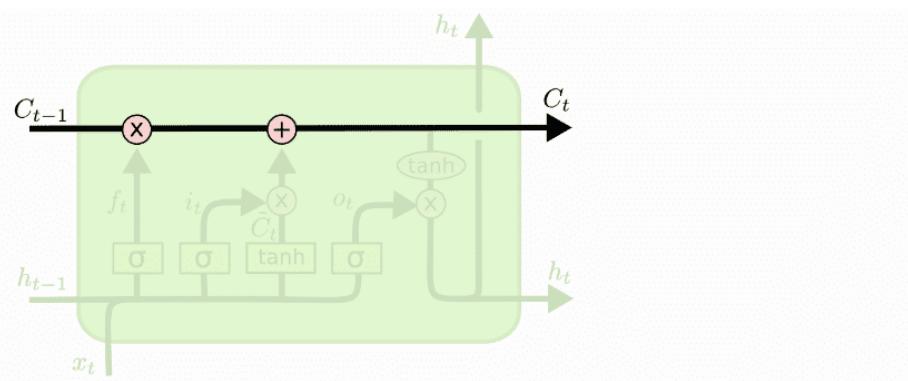
## **Redes *long short-term memory* (LSTMs)**

Este tipo de redes neuronales surge para dar solución al problema de los gradientes desaparecidos. Se tratan de una variante específica de las redes neuronales recurrentes (RNNs), diseñadas para captar relaciones de largo alcance. Desarrolladas inicialmente por Hochreiter y Schmidhuber (1997), han sido refinadas y extendidas por numerosos investigadores en estudios subsiguientes. Se han demostrado excepcionalmente efectivas en muchas aplicaciones, por eso su uso se ha generalizado hoy. La estructura de las LSTM se define en la siguiente imagen.



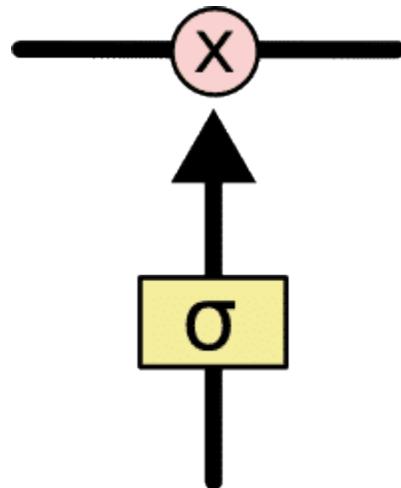
Fuente: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

La clave de las redes LSTM reside en el estado de la célula, que es la línea que se extiende horizontalmente en la parte superior de la imagen:



Fuente: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

Este estado de la célula asegura una transmisión lineal y directa de datos, minimizando las interacciones y permitiendo que la información se conserve sin cambios. Las LSTMs están equipadas con un mecanismo de puertas que regula de manera precisa la adición o eliminación de información al estado de la célula.



Fuente: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

Estas puertas, que incluyen una capa sigmoide y una operación de multiplicación, deciden la cantidad de información que debe pasar, variando de nada (valor cero) a todo (valor uno). Las LSTMs cuentan con tres de estas puertas específicas, cada una destinada a proteger y controlar el flujo de información dentro del estado de la célula, asegurando así la integridad y la relevancia de los datos procesados.



En el siguiente [link](#) puedes ver en más detalle su funcionamiento.

Entre 2013 y 2015, las LSTMs empezaron a obtener mejores resultados que otros modelos del estado del arte en tareas como reconocimiento de escritura a mano, reconocimiento de voz, traducción automática, etc. Este tipo de redes se convirtieron en el enfoque dominante para la mayoría de las tareas de NLP. Google, Apple y Facebook utilizaban LSTMs en 2016: reconocimiento de voz en los smartphones, asistentes virtuales como Siri y Alexa, el traductor de Google o la traducción automática de Facebook.

A pesar de sus buenos resultados, las LSTMs tienen algunas desventajas. ¡Vamos a revisarlas!

- Las LSTMs son computacionalmente muy costosas, ya que su estructura interna es compleja. Esto implica tiempos de entrenamiento más largos y mayor consumo de recursos
- Son difíciles de optimizar debido a su complejidad, requiriendo un conocimiento profundo del modelo. A menudo se consideran modelos de 'caja negra'.
- Tienen un alto riesgo de sobreajuste, especialmente si se entrena con conjuntos de datos pequeños.
- A pesar de ser diseñadas para manejar dependencias de largo plazo, pueden tener problemas para procesar secuencias extremadamente largas. Esto hace que información relevante pueda perderse a lo largo de la secuencia.
- Requieren de grandes cantidades de datos para generar buenos resultados.

---

**Actualmente (2019-2024), los *transformers* se han convertido en la arquitectura dominante para todas las tareas.**

# Transformers



Qualentum Lab

---

En 2017 surgió una nueva arquitectura introducida en el paper 'Attention is All You Need', que ha mostrado significantes mejoras respecto a los modelos secuencia a secuencia estudiados anteriormente (RNNs y LSTMs) en diversas tareas de NLP, tanto en calidad como en coste de entrenamiento. Esta arquitectura es la denominada 'transformer'.

---

**La arquitectura *transformer* se compone de dos partes principales: el codificador y el decodificador.**

---

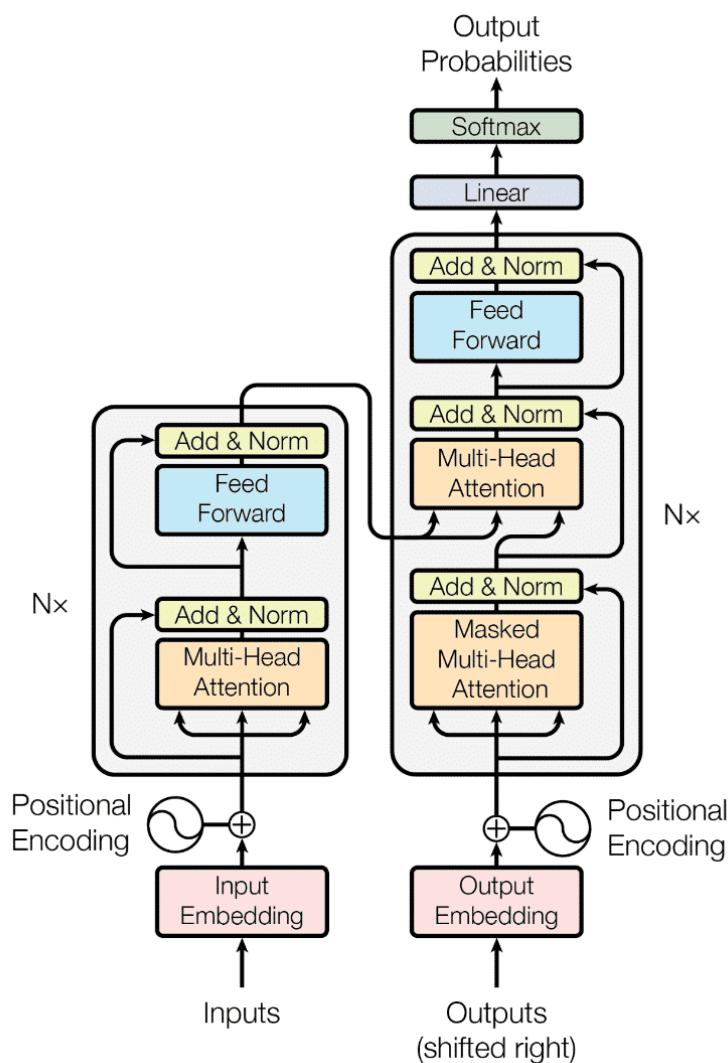
## Codificador

---

El codificador procesa la entrada de texto, aplicando una serie de capas de atención y redes neuronales para transformar la entrada en una representación vectorial rica en información

## Decodificador

El decodificador utiliza la salida del decodificador junto con la entrada previa para generar la secuencia de salida, palabra por palabra. Al igual que el codificador, el decodificador emplea mecanismos de atención que le permiten enfocarse en diferentes partes de la entrada del codificador al generar la salida.

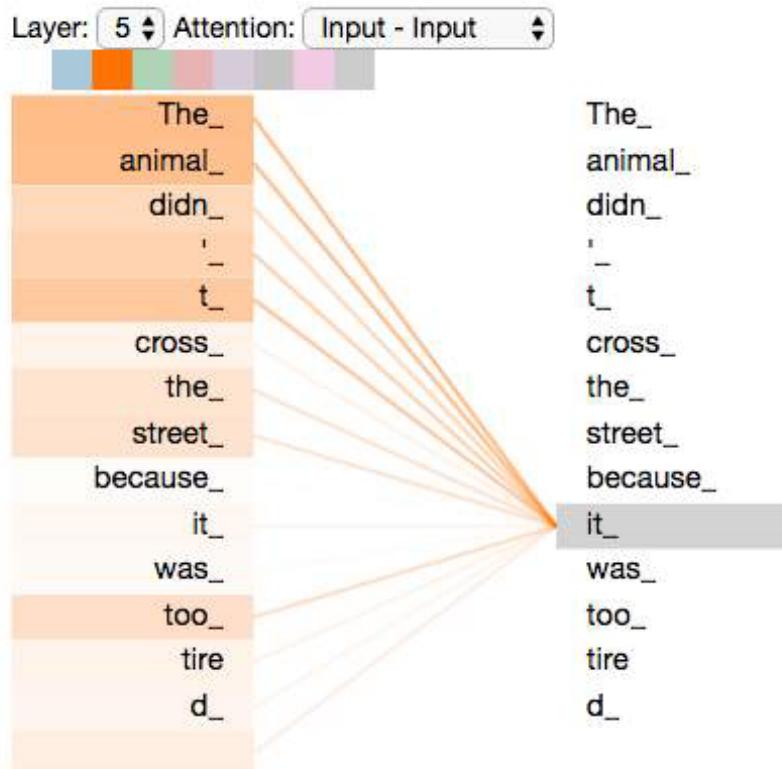


Fuente: <https://arxiv.org/pdf/1706.03762.pdf>.

La innovación clave de los transformers es el mecanismo de autoatención o self-attention, que permite al modelo ponderar la importancia de diferentes palabras en una secuencia. Así puede enfocarse en partes relevantes de la entrada para cada palabra, facilitando la captura de relaciones y dependencias sin importar la distancia entre las palabras del texto. Vamos a ver un ejemplo.

Si tenemos la frase: "The animal didn't cross the street because it was too tired".

¿A qué se refiere la palabra 'it'? ¿Se refiere a la calle o al animal? Cuando el modelo procesa la palabra 'it', la autoatención permite asociar 'it' con 'animal'. A medida que el modelo procesa cada palabra, la autoatención le permite observar otras posiciones en la secuencia de entrada en busca de pistas que puedan ayudar a codificar mejor esta palabra.



Fuente: <https://jalammar.github.io/illustrated-transformer/>.

En la arquitectura se pueden ver componentes llamados ‘multi-head attention’. ¿Entonces por qué antes hablábamos de autoatención o *self-attention*? El paper perfeccionó esta capa añadiendo un mecanismo denominado atención ‘multicabezal’ (*multi-head attention*). Esto amplía la capacidad del modelo para centrarse en distintas posiciones, permitiendo que realice múltiples operaciones de atención en paralelo.

Por último, otro componente clave en esta arquitectura son los *positional encoding*. Para compensar la ausencia de recurrencia (que sí tienen las RNNs y LSTMs), incorpora información posicional a sus entradas, asegurando que el modelo pueda considerar el orden de las palabras en la secuencia.



Fuente: <https://jalammar.github.io/illustrated-transformer/>.

¿Por qué mejoran a las RNNs y LSTMs? Vamos a poner un ejemplo.

I arrived at the **bank** after crossing the ...     ...street? ...river?

What does **bank** mean in this sentence?



RNNs

I've no idea: let's wait until I read the end



Transformer

I don't need to wait - I see all words at once!

$O(N)$  steps to process a sentence with length N

Constant number of steps to process any sentence

Fuente: [https://lena-voita.github.io/nlp\\_course/seq2seq\\_and\\_attention.html](https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html).

Estos modelos no captan el significado de la palabra 'banco' hasta haber leído la frase completa, y esto puede llevar un tiempo en secuencias largas. En cambio, en el codificador de los *transformers* las palabras son procesadas simultáneamente (procesamiento paralelo), lo cual reduce significativamente los tiempos de entrenamiento y mejora la eficiencia comparada con las arquitecturas secuenciales estudiadas con anterioridad.



En el siguiente [enlace](#) hay una guía muy detallada de cómo funcionan los transformers.

La arquitectura *transformer* ha servido de base para **el desarrollo de modelos de lenguaje preentrenados de gran escala**, como como BERT (*bidirectional encoder representations from transformers*) o GPT (*generative pretrained transformer*).

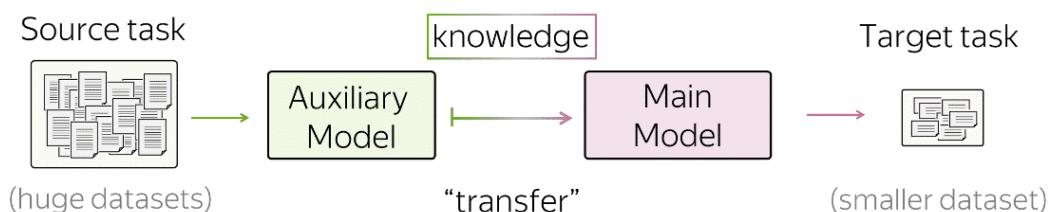
# Aprendizaje por transferencia (transfer learning)



Antes de los famosos modelos de lenguaje de gran tamaño (LLMs), el aprendizaje por transferencia era probablemente el área de NLP más popular. Si te suenan modelos como ELMo o BERT, después de este apartado entenderás por qué.

**El concepto fundamental detrás del aprendizaje por transferencia consiste en trasladar el conocimiento aprendido de una tarea o modelo específico hacia otro distinto.**

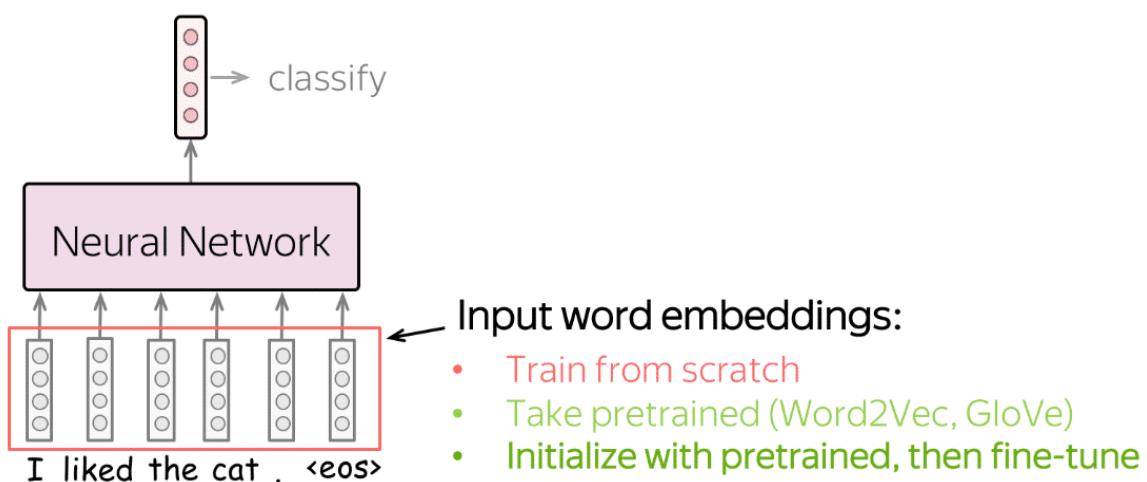
Esto es útil cuando nos enfrentamos a una tarea para la que no tenemos muchos datos. Por otro lado, podemos disponer de datos para una tarea diferente, cuya recolección resulte más sencilla (por ejemplo, para entrenar modelos de lenguaje no se requieren datos etiquetados, sino simplemente textos en formato plano).



Fuente: [https://lena-voita.github.io/nlp\\_course/transfer\\_learning.html](https://lena-voita.github.io/nlp_course/transfer_learning.html).

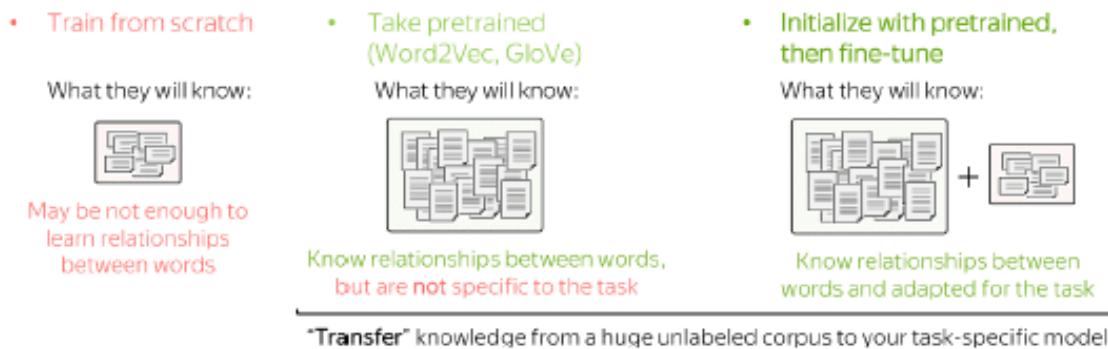
Al principio del fastbook hablábamos de los *word embeddings*, pues usar *word embeddings* preentrenados puede ayudar mucho a resolver tareas de NLP. Cuando nos enfrentamos a una tarea, por ejemplo, de clasificación de texto, tenemos tres opciones para obtener los *embeddings* de nuestro modelo:

- Entrenar los *embeddings* desde cero.
- Elegir un modelo preentrenado (word2vec, GloVe) y fijar los *embeddings* (usarlos como vectores estáticos).
- Inicializar con *embeddings* preentrenados y entrenar los *embeddings* con la red (lo que llamamos ‘fine-tune’).



Fuente: [https://lena-voita.github.io/nlp\\_course/transfer\\_learning.html](https://lena-voita.github.io/nlp_course/transfer_learning.html).

Si entrenamos los *embeddings* desde cero, el modelo estará limitado a entender únicamente la información presente en los datos de clasificación. Sin embargo, al optar por *embeddings* que ya han sido entrenados, el modelo completo tendrá acceso a un amplio corpus de conocimiento.



Fuente: [https://lena-voita.github.io/nlp\\_course/transfer\\_learning.html](https://lena-voita.github.io/nlp_course/transfer_learning.html).

Esto es un ejemplo de aprendizaje por transferencia: a través de los *embeddings* preentrenados estamos ‘trasladando’ el conocimiento obtenido de sus datos de entrenamiento originales hacia nuestro modelo particular diseñado para una tarea específica.



En el siguiente [link](#) encontrarás más información sobre este tema.

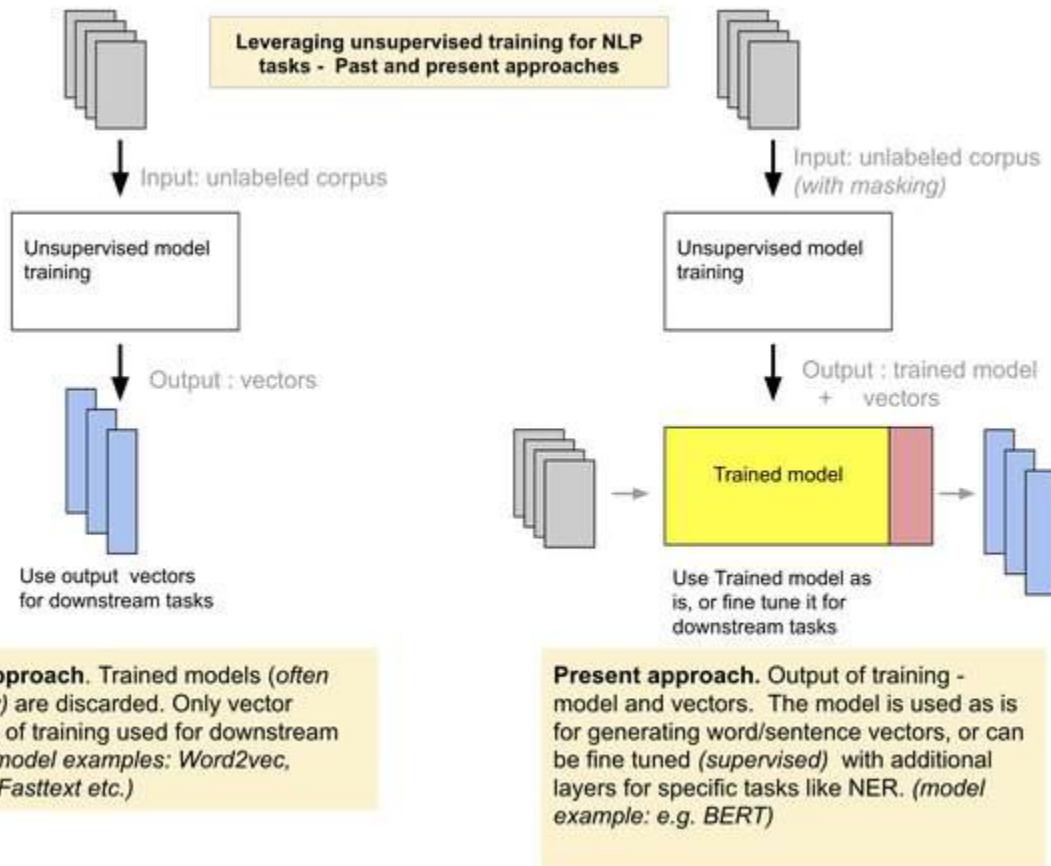
# Modelos preentrenados (pretrained models)



---

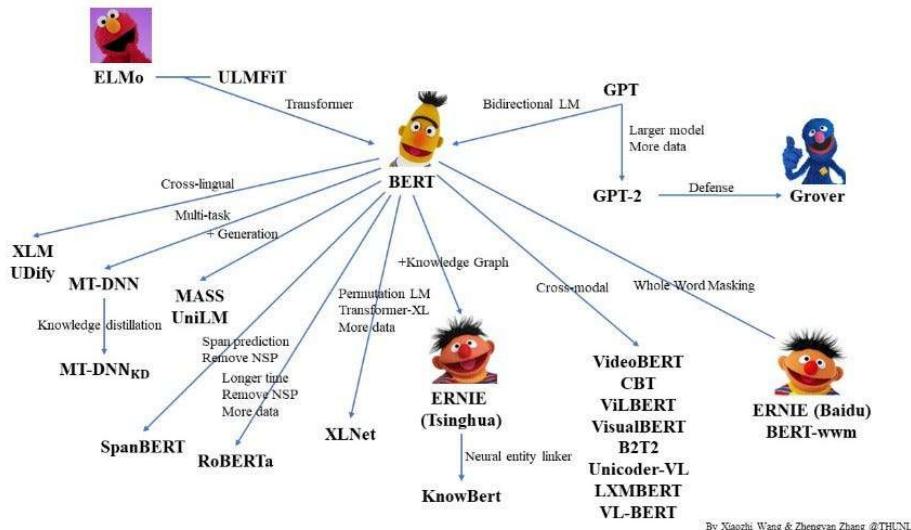
Lo que hemos aprendido en el apartado anterior con los *word embeddings* se puede aplicar a los modelos preentrenados. La transición de *word embeddings* a modelos de estado del arte actuales se puede explicar con dos ideas:

- lo que se representa, es decir, pasamos de representar palabras de forma aislada a representar palabras dentro de su contexto específico (de word2vec/GloVe a ELMo);
- y cómo se aplican estos modelos a tareas posteriores. Evolucionamos de simplemente reemplazar los *word embeddings* en modelos diseñados para tareas particulares a reemplazar los modelos completos de esas tareas específicas (de ELMo a GPT/BERT).
- Welcome diversions. The most rewarding adventures often start with an unexpected detour. Perhaps that distraction will guide you onward.



Fuente: <https://www.quora.com/What-were-the-most-significant-Natural-Language-Processing-advances-in-2018>.

En este apartado veremos dos modelos preentrenados que hacen referencia a cada una de estas dos ideas: ELMo y BERT. Como se puede ver en la siguiente imagen, existen muchas variaciones de estos modelos.



Fuente: <https://towardsdatascience.com/pre-trained-language-models-simplified-b8ec80c62217>.

## ELMo (*embeddings from language models*)

*Embeddings from language models*, mejor conocido por sus siglas 'ELMo' ([2018](#)) representa la primera idea: en vez de representar palabras aisladas, podemos aprender a representar palabras dentro del contexto en el que se usan.

Por ejemplo, si utilizamos representaciones como GloVe, la palabra palo, 'stick', tendrá múltiples significados dependiendo del contexto en el que se use. Es por ello por lo que se crearon los **word embeddings contextualizados**, para capturar el significado de la palabra en ese contexto además de otra información contextual.



Fuente: <https://jalammar.github.io/illustrated-bert/>.

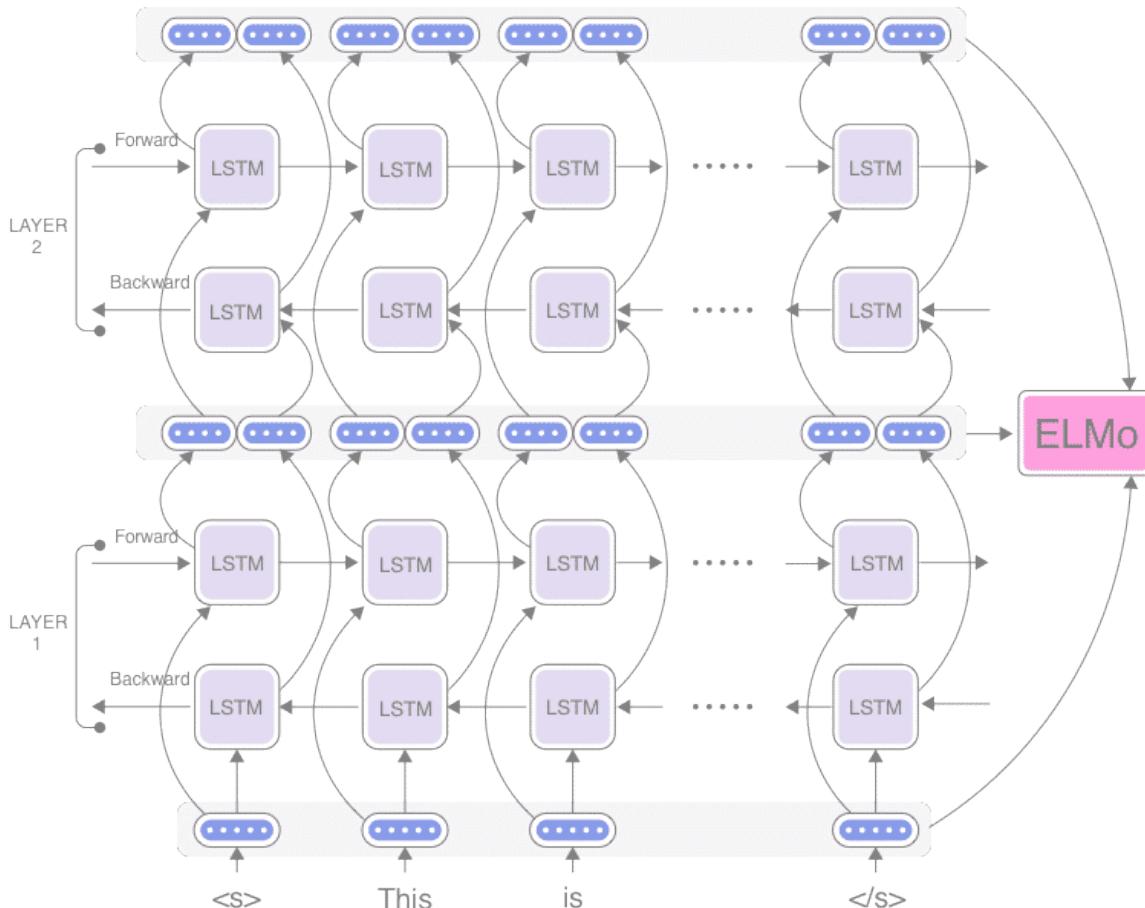
A diferencia los métodos de *word embedding* tradicionales, ELMo es dinámico, lo que viene a decir que sus *embeddings* cambian dependiendo del contexto incluso cuando la palabra es la misma. Esto lo consigue reemplazando los *word embeddings* (GloVe) con *embeddings* de un modelo de lenguaje (LM) preentrenado. De ahí viene el 'LM' en el nombre ELMo. Con esto consiguieron una gran mejora para diversas tareas como respuesta a preguntas, análisis de sentimientos, etc.

---

**La arquitectura de ELMo es muy simple: consiste en una combinación dos LMs LSTM de dos capas en ambas direcciones, uno hacia delante y hacia atrás (bidireccional).**

---

Los dos modelos se utilizan para leer la frase en ambas direcciones y así obtener contexto de izquierda y derecha.



Fuente: <http://www.realworldnlpbook.com/blog/improving-sentiment-analyzer-using-elmo.html>.



En los siguientes links hay una explicación más detallada de ELMo: <https://jalammar.github.io/illustrated-bert/> o <http://www.realworldnlpbook.com/blog/improving-sentiment-analyzer-using-elmo.html>. ¡Revísalos!

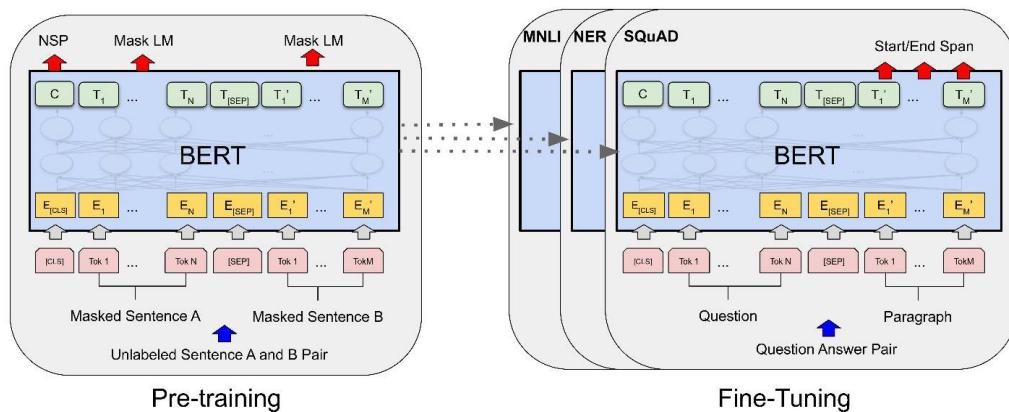
## BERT (*bidirectional encoder representations from transformers*)

El siguiente modelo que vamos a estudiar, bidirectional encoder representations from transformers 'BERT' (2018), es muy diferente a las aproximaciones anteriores que hemos visto.

BERT reemplaza el modelo entero, no como ELMo que reemplaza solo los *word embeddings*.

Antes teníamos una arquitectura de modelo específica para cada tarea posterior. Esto significa que para cada tarea tenemos que usar un modelo específico (respuesta a preguntas, análisis de sentimientos, etc.). Ahora tenemos un modelo único que puede ser *fine-tuneado* para todas las tareas.

La arquitectura de BERT es también muy sencilla y ya sabes cómo funciona: es solo el codificador de la arquitectura *transformer*.



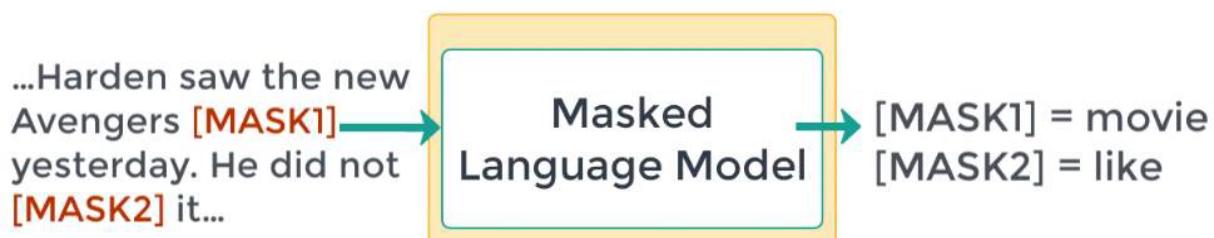
Fuente: <https://arxiv.org/pdf/1810.04805.pdf>.

Al contrario de los modelos direccionales, que leen el texto de entrada secuencialmente (de izqda. a dcha., o viceversa), el codificador de *transformer* lee toda la secuencia de palabras a la vez (en paralelo). Por eso se considera bidireccional, aunque es más correcto decir que es no direccional. Esta característica permite al modelo aprender el contexto a la izquierda y derecha de la palabra.

Al entrenar modelos de lenguaje, el desafío es definir el objetivo de la predicción. Muchos modelos predicen la siguiente palabra en una secuencia, lo cual limita el aprendizaje del contexto. Para superar este reto, BERT utiliza dos estrategias de entrenamiento: el LM enmascarado y la predicción de la siguiente frase.

- **LM enmascarado (MLM)**

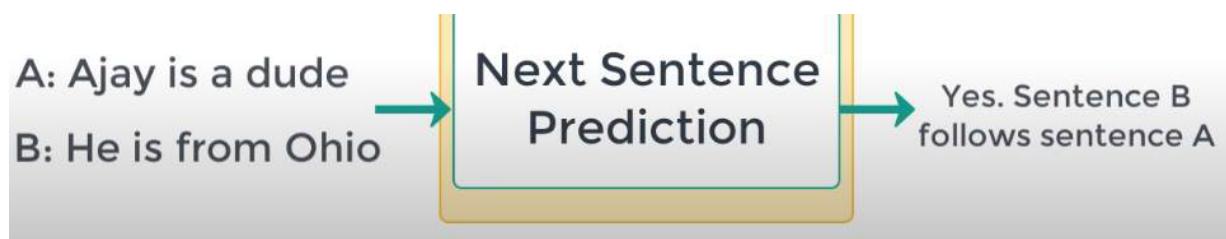
Antes de alimentar al modelo con secuencias de palabras, el 15% de las palabras son sustituidas por el token [MASK]. El modelo entonces intenta predecir el valor original de estas palabras enmascaradas, basándose en el contexto dado por otras palabras no enmascaradas de la secuencia.



Fuente: <https://www.youtube.com/watch?v=BGKumhtlqLA>.

- **Predicción de la siguiente frase (NSP)**

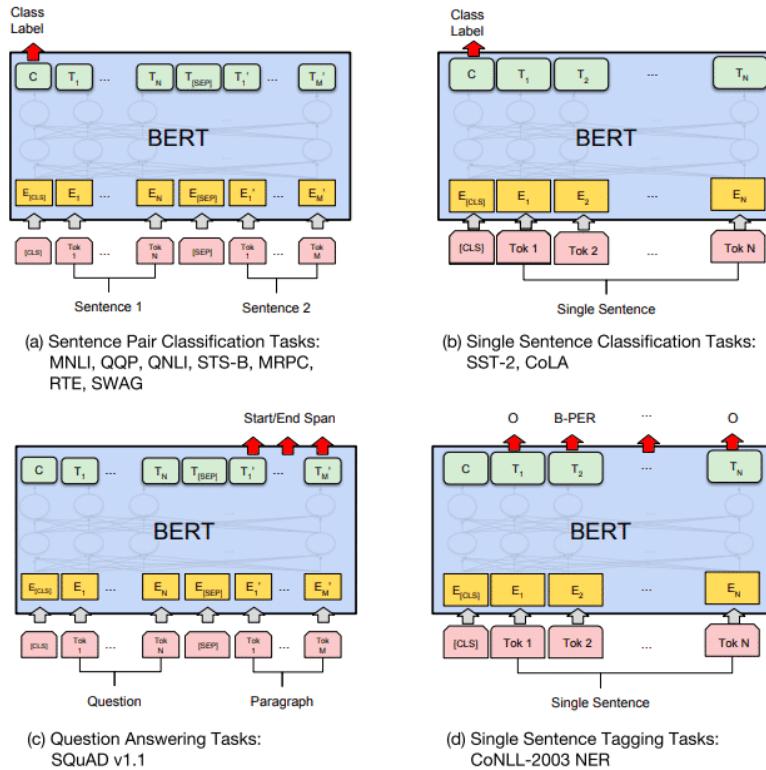
El modelo recibe pares de frases como entrada y aprende a predecir si la segunda frase del par es la frase siguiente en el documento original. Durante el entrenamiento, el 50% de las entradas son un par de frases en el que la segunda constituye la siguiente en el documento original, mientras que en el 50% restante se elige una frase aleatoria como segunda frase.



Fuente: <https://www.youtube.com/watch?v=BGKumhtlqLA>.

Se puede hacer *finetuning* de BERT para diversas tareas, por ejemplo:

- para la clasificación de una sola frase;
- en la clasificación de pares de frases;
- para la respuesta a preguntas (*question answering*);
- y en el etiquetado de una sola frase.



Fuente: <https://arxiv.org/pdf/1810.04805.pdf>.



Puedes aprender más de este modelo en los recursos extras accesibles desde los siguientes enlaces:

<https://jalammar.github.io/illustrated-bert/> y  
<https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>.

# Técnicas de optimización de entrenamientos



Si recordamos, hay dos problemas comunes en los modelos de deep learning: el ‘underfitting’ y el ‘overfitting’, lo que puede conllevar a resultados subóptimos. En este apartado estudiaremos de qué maneras podemos ajustar un modelo para mejorar su rendimiento.

## Sesgo y varianza

A la hora de **medir el rendimiento de un modelo** hay que fijarse en el sesgo, la varianza y el equilibrio entre ellos.

### SESGO

### VARIANZA

El sesgo es la diferencia entre la predicción media de nuestro modelo y el valor correcto que se quiere predecir. Un sesgo alto significa que el modelo no se ajusta a los datos de entrenamiento, conocido como subajuste (*underfitting*). Esto se traduce en un error alto en los datos de entrenamiento y test.

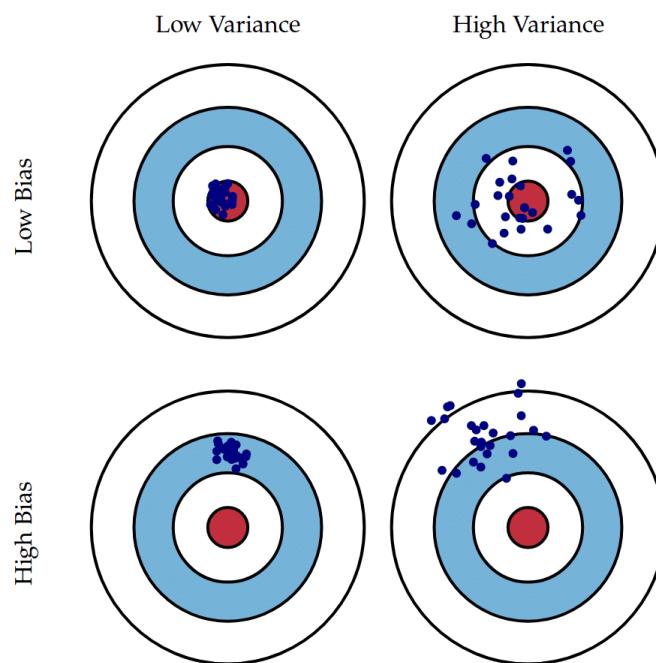
**SESGO****VARIANZA**

La varianza es la variabilidad del modelo al ajustarse a diferentes conjuntos de datos. Un modelo con alta varianza significa que se ajusta demasiado a los datos de entrenamiento, lo que lleva a un sobreajuste (*overfitting*).

---

**Un modelo demasiado simple tendrá alto sesgo y baja varianza, mientras que un modelo demasiado complejo tendrá bajo sesgo y alta varianza.**

---



Graphical illustration of bias and variance. Fuente: <https://scott.fortmann-rothe.com/docs/BiasVariance.html>.

---

## Hiperparámetros

Antes de elegir una estrategia para mejorar el rendimiento de un modelo, primero hay que diferenciar entre parámetro e hiperparámetro.

### Parámetro

Su valor se obtiene automáticamente en el proceso de entrenamiento.

### Hiperparámetro

Su valor se fija manualmente antes de empezar el proceso de entrenamiento.

Algunos hiperparámetros que especifican la estructura de la red neuronal son los siguientes, ¡recuerda!

- El número de capas.
- El número de unidades en las capas ocultas.

- Las funciones de activación.
- Las funciones de error.
- El número de iteraciones.
- La ratio de aprendizaje.
- El tamaño del batch.
- Otros:
  - El método de inicialización (He, Xavier...).
  - El método de regularización (Dropout, L2...).
  - El método de optimización (Momentum, Adam...).

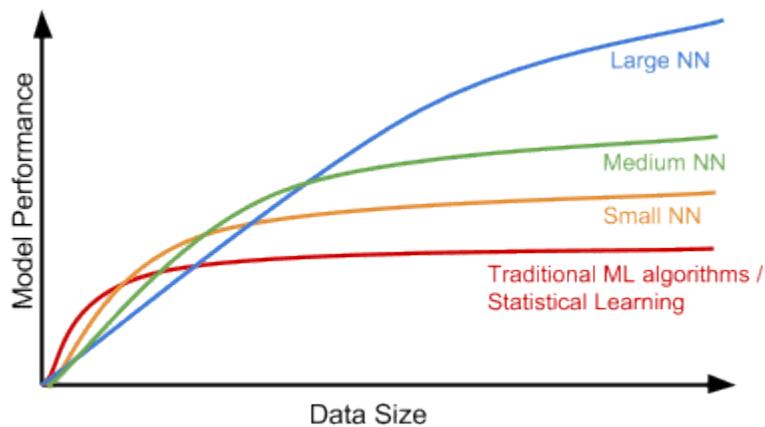
## Estrategias para mejorar el rendimiento de un modelo

El rendimiento de un modelo puede incrementarse con diferentes estrategias:

- Más complejidad de la red.
- Más datos.

Esto conlleva algunas limitaciones:

- La potencia computacional.
- La disponibilidad de datos etiquetados.



Fuente: <https://lilianweng.github.io/posts/2017-06-21-overview/>.

Estas son algunas de las estrategias para **reducir el sesgo**:

- Añadir complejidad al modelo.
- Entrenar más tiempo.
- Elegir un mejor algoritmo de optimización.

Y estas son dos estrategias para **reducir la varianza**:

- Entrenar con más datos.
- Añadir regularización.

## **Elección de hiperparámetros básicos**

---

El tamaño del batch es el número de ejemplos procesados para calcular la función de coste justo antes de actualizar los pesos.

Sus posibles valores pueden ser:

DESCENSO DE GRADIENTE BATCH	DESCENSO DE GRADIENTE ESTOCÁSTICO	DESCENSO DE GRADIENTE MINI-BATCH
Tamaño del batch = nº total de ejemplos.		

**DESCENSO DE GRADIENTE  
BATCH**

**DESCENSO DE GRADIENTE  
ESTOCÁSTICO**

**DESCENSO DE GRADIENTE  
MINI-BATCH**

Tamaño del batch = 1.

**DESCENSO DE GRADIENTE  
BATCH**

**DESCENSO DE GRADIENTE  
ESTOCÁSTICO**

**DESCENSO DE GRADIENTE  
MINI-BATCH**

Tamaño del batch =  $1 < k < \text{nº total de ejemplos}$ .

A la hora de elegir el tamaño del *batch*, debemos tener en cuenta estas consideraciones:

1

Si el conjunto de entrenamiento es pequeño ( $\text{nº total de ejemplos} < 2000$ ), el tamaño del batch debe ser igual al  $\text{nº total de ejemplos}$

2

Tenemos que asegurarnos de que el mini-batch cabe en memoria CPU/GPU.

3

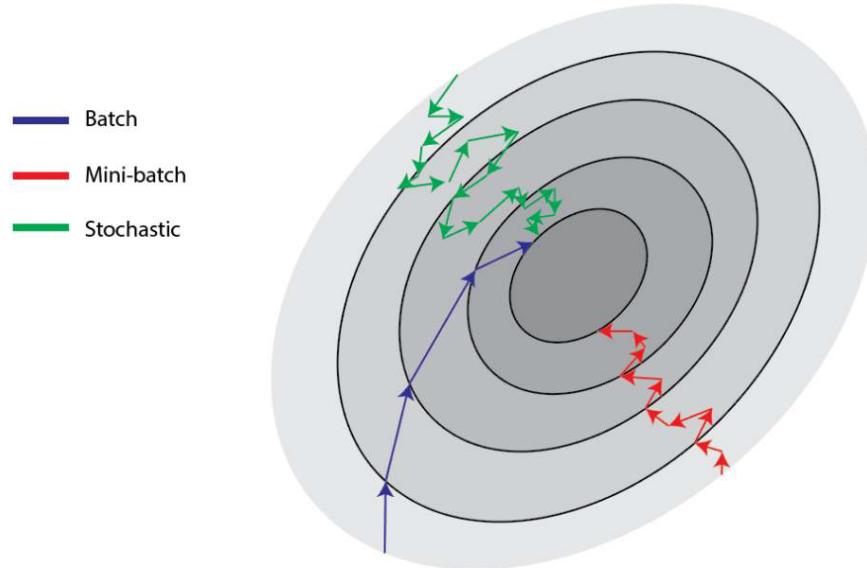
Recordemos que algunos tamaños de batch típicos son 64, 128, 256, 512, 1024...

4

La función de coste puede ser vista como una superficie representada por líneas de contorno.

5

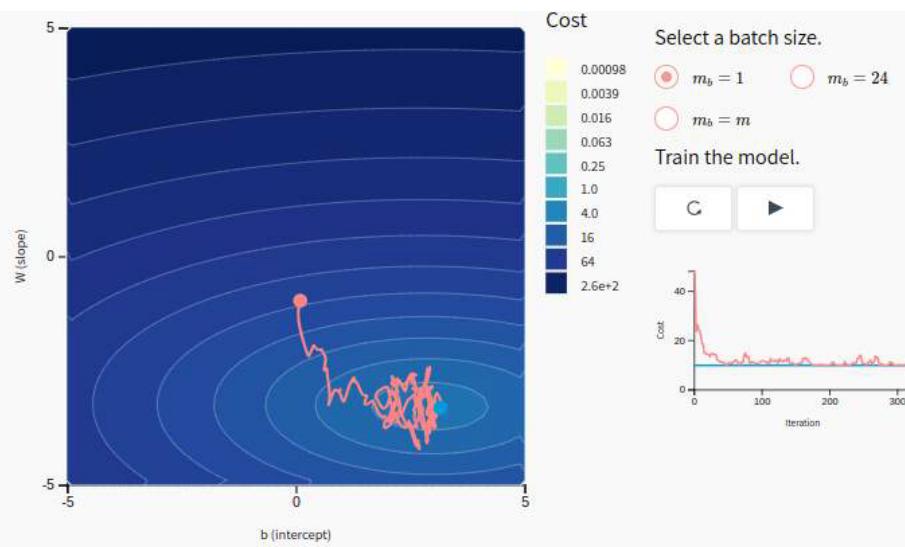
El proceso de entrenamiento intenta alcanzar el valor mínimo en un proceso iterativo.



Fuente: <https://www.baeldung.com/cs/learning-rate-batch-size>.

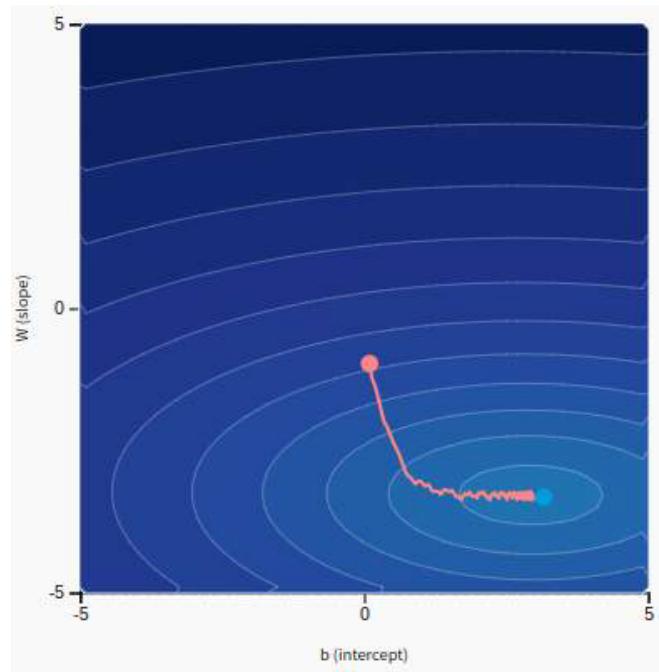
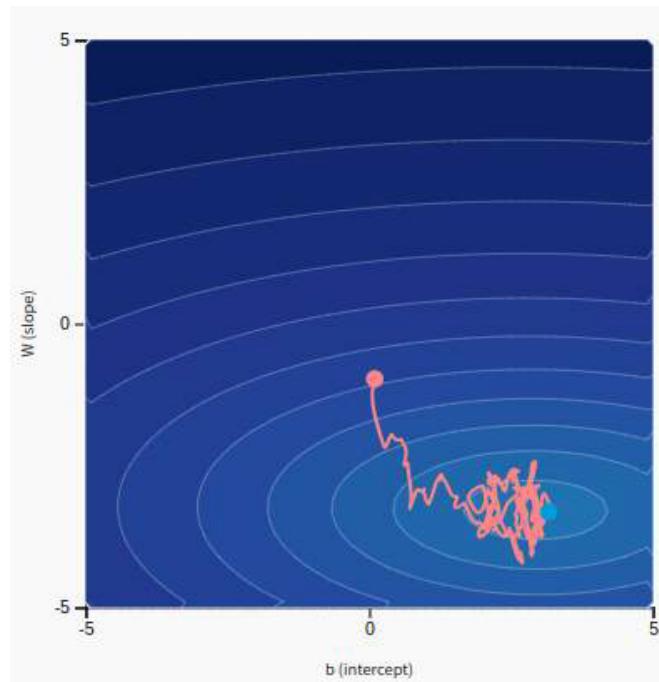


En el siguiente [enlace](#) se explica de manera intuitiva cómo optimizar un modelo:



Fuente: <https://www.deeplearning.ai/ai-notes/optimization/index.html>.

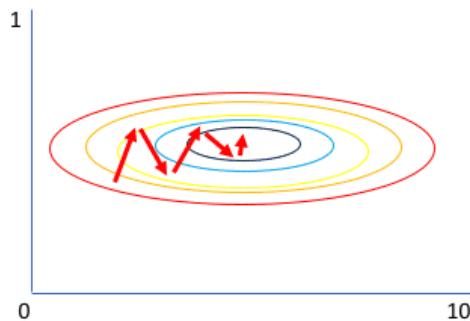
No olvidemos tampoco que cuanto más grande es el tamaño del batch, más preciso es el gradiente del coste. En las siguientes imágenes vemos la diferencia entre un tamaño de batch de 1 (izqda.) y 24 (dcha.).



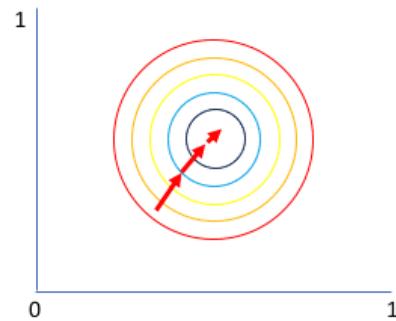
## La elección de hiperparámetros avanzados

El proceso de entrenamiento **intenta alcanzar el valor mínimo en un proceso iterativo**. Los datos de entrada con rango de valores muy diferentes producen superficies alargadas para la función de coste, lo cual puede generar una convergencia ineficiente al valor mínimo.

### Why normalize?



Gradient of larger parameter  
dominates the update



Both parameters can be  
updated in equal proportions

Fuente: <https://www.jeremyjordan.me/batch-normalization/>.

Los datos de entrada pueden ser normalizados usando ‘min-max scaling’:

$$X = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Así rescaldamos los valores entre el rango [0,1]. El problema es que los valores atípicos afectan a este tipo de escalado. Por ello, podemos normalizar los datos usando *standardization*:

- $X = X - \mu / \sigma$

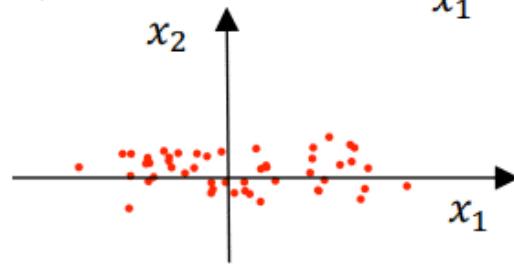
De esta manera, los valores tienen media cero y desviación estándar 1. A continuación, vemos una interpretación gráfica:

Initial values:



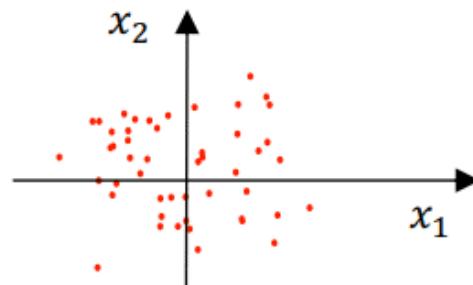
Values are shifted:

$$1. X \leftarrow X - \mu$$



Values are scaled:

$$2. X \leftarrow X / \sigma^2$$



Fuente: Andrew Ng, (2017). Neural Networks and Deep Learning. Coursera.

## Inicialización

La inicialización de los pesos del modelo sirve para definir los valores iniciales de los parámetros en un modelo antes de entrenar sobre un conjunto de datos.

Algunos métodos son 'He' y 'Xavier'; y estos son algunos de sus beneficios:

- Convergencia más rápida durante el entrenamiento.
- Mejora la capacidad del modelo para alcanzar un mínimo global en la función de pérdida.
- Ayuda a mantener la varianza de las activaciones a través de las capas.



Un recurso para entender la inicialización:

<https://www.deeplearning.ai/ai-notes/initialization/index.html>.

## Normalización del *batch*

La normalización del *batch* es una técnica de regularización que normaliza la salida de la capa anterior. Este método añade complejidad a la red neuronal, y en cuanto a sus ventajas destacamos dos de ellas: **la convergencia más rápida del entrenamiento y la reducción del problema de desvanecimiento o explosión de gradientes.**

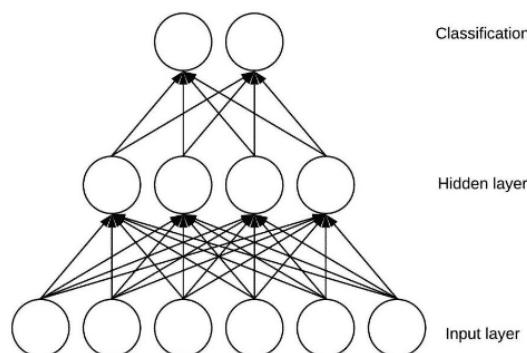


En el siguiente [link](#), puedes ver una representación visual de la normalización del *batch*.

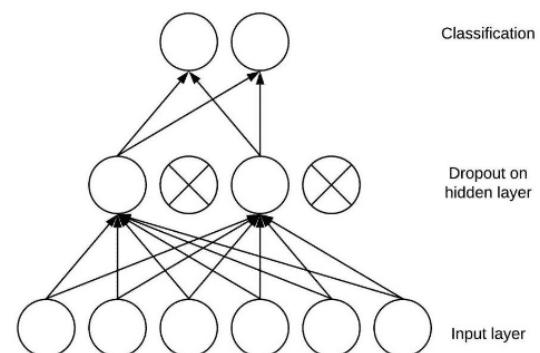
## Regularización

La regularización es un conjunto de técnicas para prevenir el sobreajuste u *overfitting*. Este método ayuda a mejorar la capacidad de generalización del modelo, evitando que se vuelva excesivamente complejo. A continuación, vamos a estudiar sus técnicas.

- **Dropout:** consiste en desactivar aleatoriamente un porcentaje de neuronas en las capas de red durante el entrenamiento. Esto fomenta una mejor distribución del aprendizaje a través de toda la red.



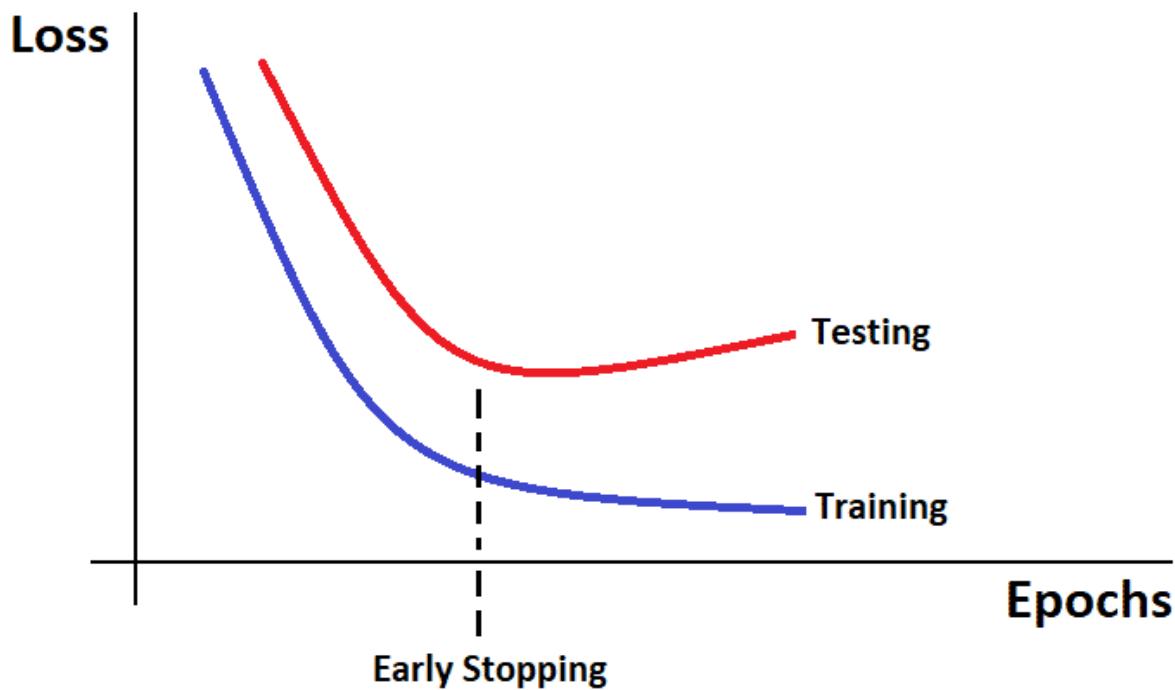
**Without Dropout**



**With Dropout**

Fuente: <https://wenkangwei.github.io/2020/11/13/DL-DropOut/>.

- **L1 y L2 (lasso y ridge):** estas técnicas agregan un término de penalización al coste del modelo basados en los valores de los pesos. En el caso de L1 o lasso, tiende a reducir el valor de los pesos hacia cero. L2 o ridge, en cambio, fuerza a que los pesos sean pequeños, pero no haciéndolos exactamente cero. L1 es más robusto que L2, aunque L2 requiere menos recursos computacionales. L1 tiende a generar modelos más dispersos, frente a L2 que tiende a distribuir los pesos de manera más uniforme.
- **Early stopping:** consiste en detener el entrenamiento antes de que el modelo haga *overfitting*. Esto se hace cuando el rendimiento del modelo deja de mejorar, observando el rendimiento sobre el conjunto de validación.



Fuente: <https://pub.towardsai.net/keras-earlystopping-callback-to-train-the-neural-networks-perfectly-2a3f865148f7>.

## Optimización

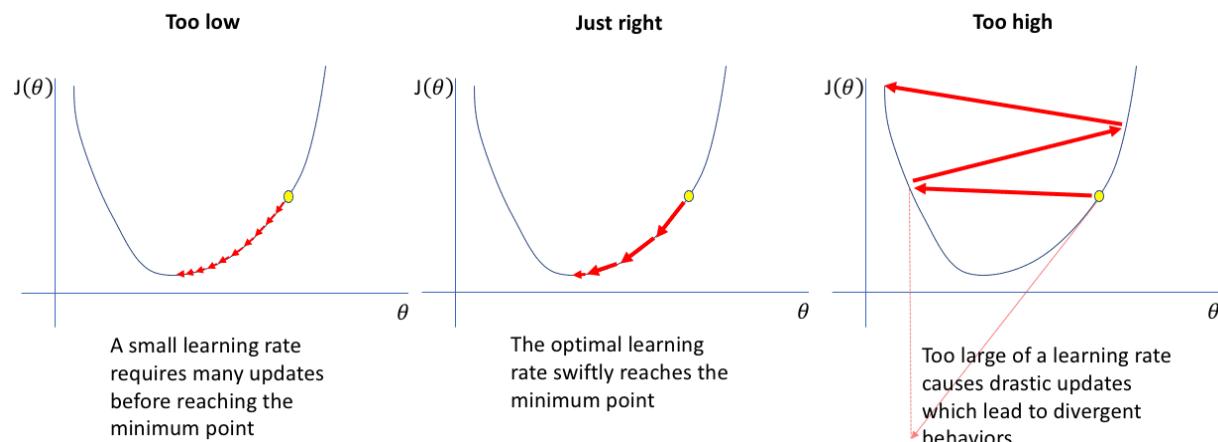
---

La optimización se refiere al proceso de ajustar los hiperparámetros de una red neuronal para minimizar la función de pérdida o coste.

El objetivo es encontrar el conjunto de parámetros que haga que el modelo aprenda de manera eficaz. Veamos algunos de los algoritmos principales, ¡apunta!



**Decaimiento de la tasa de aprendizaje (*learning rate decay*)**: el objetivo principal es mejorar la convergencia del modelo al empezar con una tasa de aprendizaje relativamente alta y luego ir disminuyéndola gradualmente para permitir ajustes más finos y evitar sobrepasar el mínimo óptimo de la función de pérdida.



Fuente: <https://www.jeremyjordan.me/nn-learning-rate/>.

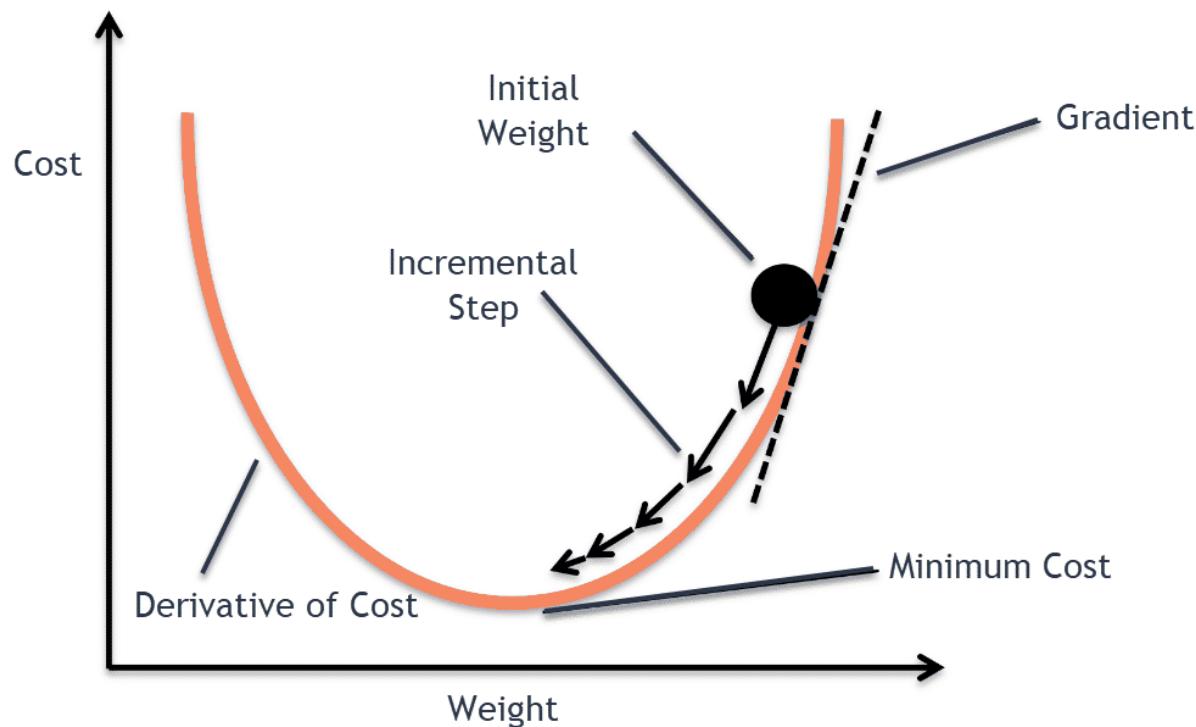
---



En el siguiente [link](#), descubrirás un post con una explicación detallada de cómo encontrar un *learning rate* óptimo.



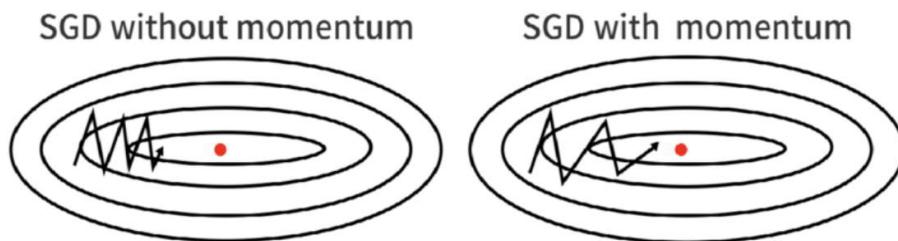
**Descenso de gradiente estocástico (SGD):** en el descenso de gradiente tradicional, el gradiente se calcula utilizando todo el conjunto de datos de entrenamiento. En cambio, en el SGD, el gradiente se calcula y los parámetros se actualizan para cada ejemplo de entrenamiento o para pequeños subconjuntos (llamados 'minibatch'). De esta manera el modelo se actualiza muchas veces en cada epoch, lo que puede llevar a una convergencia más rápida.



Fuente: [https://medium.com/@divakar\\_239/stochastic-vs-batch-gradient-descent-8820568eada1](https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1).



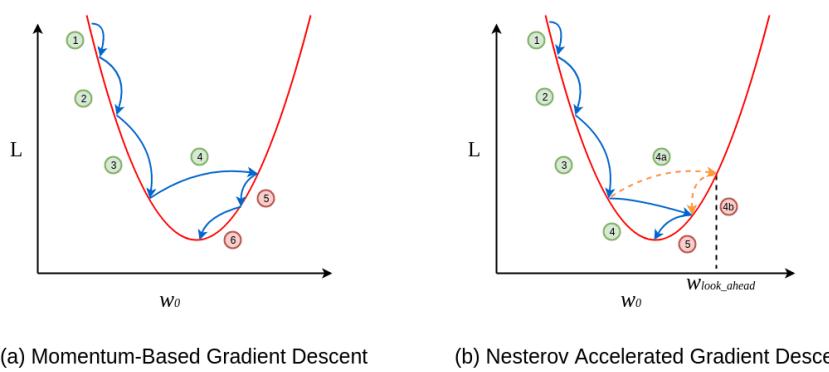
**Momentum:** inspirado en la física y la noción de momento (o impulso), esta técnica ayuda a que el algoritmo de optimización converja más rápidamente hacia el mínimo global de la función de pérdida, evitando quedar atrapado en mínimos locales. La idea es incorporar una fracción del vector de actualización de los pesos de la iteración anterior en la actualización actual.



Fuente: <https://paperswithcode.com/method/sgd-with-momentum>.



**Gradiente acelerado nesterov (NAG):** es una variante refinada de la técnica *momentum*. La principal innovación es la forma en que se calcula el gradiente: mientras que *momentum* primero ajusta los parámetros basándose en la acumulación previa de gradientes y luego calcula el gradiente, NAG ajusta primero los parámetros de manera provisional, y después calcula el gradiente en esa posición provisional. Esto hace que el método sea más sensible a la superficie de la función de pérdida.



$$\textcolor{green}{\bullet} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Positive}(+)} \quad \textcolor{red}{\bullet} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Negative}(-)}$$

Fuente: <https://towardsdatascience.com/learning-parameters-part-2-a190bef2d12>.

- RMSprop:** la idea es modificar la tasa de aprendizaje para cada parámetro del modelo de manera individual, dividiendo la tasa de aprendizaje por un promedio móvil del cuadrado de los gradientes. Esto ayuda a mitigar el problema del SGD, donde una tasa de aprendizaje única se aplica a todos los parámetros del modelo.
- Adam:** este algoritmo combina los enfoques RMSprop y *momentum*. Es uno de los optimizadores más populares. Adam mantiene dos estimaciones de momentos para cada parámetro del modelo: el primero es un promedio móvil exponencial de los gradientes pasados (similar al *momentum*), y el segundo es un promedio móvil exponencial de los cuadrados de los gradientes pasados (similar a RMSprop). Adam obtiene la velocidad del *momentum* y la capacidad de adaptar los gradientes en diferentes direcciones del RMSProp, haciéndolo más efectivo.

## Resumen esquemático de estos algoritmos

- Decaimiento de la tasa de aprendizaje (*learning rate decay*) → la tasa de aprendizaje disminuye a medida que avanza el entrenamiento.
- Descenso de gradiente estocástico (SGD) → los parámetros se actualizan usando subconjuntos aleatorios de datos.
- *Momentum* → crea movimiento con la inercia.
- Gradiente acelerado Nesterov (NAG) → corrige el impulso (momento) con un vector más eficaz.
- RMSprop → reduce el paso en la dirección de gradientes mayores.
- Adam → Combina *momentum* y RMSprop.



Para ver animaciones visuales de varios algoritmos de optimización, consulta estos enlaces:  
<https://www.denizyuret.com/2015/03/alec-radfords-animations-for.html> y <https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c>. Y en este [link](#) puedes encontrar una guía sobre optimizadores muy recomendable.

# Conclusión



---

En este fastbook empezábamos aprendiendo qué es el procesamiento de lenguaje natural (NLP) y cuáles son sus casos de uso. Primero hemos explicado cómo son transformados los textos en valores numéricos para que sean entendidos por las máquinas y los diferentes métodos para crear representaciones vectoriales de palabras y documentos, desde los más básicos, que no utilizan redes neuronales (*one-hot vectors, bag of words, TF-IDF*) hasta los que sí (*word2vec, GloVe*).

A continuación, hemos estudiado los modelos secuenciales (RNNs y LSTMs) y por qué mejoran a los modelos estudiados con anterioridad. Vimos la arquitectura transformer y por qué supuso una revolución en el mundo de la IA, en concreto, en el campo del NLP. Por último, aprendimos qué es el aprendizaje por transferencia y qué son los modelos preentrenados, poniendo ejemplos de dos modelos de ellos (ELMo y BERT).

En el último bloque, hemos querido abordar la optimización de los modelos de deep learning para mejorar su rendimiento. Hemos repasado cómo se evalúan los modelos a través del sesgo y la varianza, por qué hay que encontrar un equilibrio entre ellos y cuáles son las diferentes estrategias para reducirlos. Antes de conocer estas estrategias hemos visto lo que son los hiperparámetros y cuáles componen los modelos de redes neuronales. Por último, hemos aprendido qué hiperparámetros escoger para optimizar el rendimiento de los modelos, desde los más básicos hasta los más avanzados.

Con esta travesía por el NPL ya dispones de la información necesaria para poner en práctica todo lo aprendido.

# Referencias



Qualentum Lab

- 
- <https://www.deeplearning.ai/resources/natural-language-processing/>
  - [https://lena-voita.github.io/nlp\\_course.html](https://lena-voita.github.io/nlp_course.html)
  - <https://web.stanford.edu/class/cs224n/>
  - <https://neptune.ai/blog/vectorization-techniques-in-nlp-guide>
  - <https://jalammar.github.io/illustrated-word2vec/>
  - <https://blog.acolyer.org/2020/12/08/bias-in-word-embeddings/>
  - <https://jalammar.github.io/illustrated-transformer/>
  - <http://www.realworldnlpbook.com/blog/improving-sentiment-analyzer-using-elmo.html>
  - <https://jalammar.github.io/illustrated-bert/>
  - <https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229>
  - <https://www.deeplearning.ai/ai-notes/index.html>

¡Enhорabuena! Fastbook superado



[Qualentum.com](http://Qualentum.com)