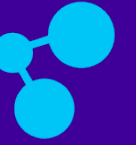




Inteligencia Artificial para profesionales TIC

Introducción a la implementación de LLMs



Índice

01. Introducción

02. Embeddings

03. RNN

04. Transformers

05. Similitud semántica

06. RAG y fine-tuning



1. Introducción

De embeddings a transformers

- Hemos visto qué es un modelo de lenguaje y una idea básica sobre cómo se entrena (diferentes etapas, la primera autosupervisada)
- Vamos a ver conocer más detalles técnicos sobre las NN que los pueden implementar
- Los **Embeddings** han revolucionado la forma en que manejamos los datos en el aprendizaje automático, particularmente en el dominio del procesamiento del lenguaje natural (NLP).
- Como **representaciones vectoriales de alta dimensión**, nos permiten convertir datos complejos como palabras, oraciones o incluso documentos completos en una forma que los algoritmos de aprendizaje automático pueden procesar.
- En NLP, los embeddings de palabras han dado lugar a un salto significativo en la realización de diversas tareas al capturar el significado semántico y las relaciones entre las palabras.
- Permiten a los modelos entender que palabras como "rey" y "reina" se relacionan de manera similar a "hombre" y "mujer".
- En cierto modo, los embeddings actúan como un **punto** entre **los datos sin procesar y no estructurados y el mundo estructurado de los algoritmos de aprendizaje automático**.



1. Introducción

De embeddings a transformers

- La arquitectura de red neuronal **Transformer** (2017 Google), originalmente propuesta para tareas de NLP, ha experimentado una creciente adopción y éxito en innumerables contextos.
- Esta arquitectura, con su mecanismo de **autoatención**, permite a los modelos **comprender el contexto de cada palabra en una oración y sopesar la importancia de cada palabra al predecir la siguiente**. Los transformers han logrado resultados impactantes en tareas de referencia en NLP.
- Curiosamente, la aplicación de Transformers **no se ha limitado solo a la NLP**.
- Ahora también están avanzando en tareas de **visión** por computadora, con modelos como **Vision Transformer (ViT)** que aplican la arquitectura del transformer directamente a los píxeles de la imagen, lo que muestra el impacto de gran alcance y la versatilidad de estos modelos en diferentes aplicaciones.



1. Introducción

NLP (Natural Language Processing)

- Nuestro **lenguaje humano es realmente complejo**. Lleno de ambigüedades, excepciones, irregularidades y muchos otros elementos que no son compatibles con las reglas estrictas y fijas requeridas que necesita una CPU.
- Desde el comienzo de la era de la informática, hemos estado tratando de comunicarnos con los dispositivos en lenguaje natural, y nacieron áreas específicas, como el Procesamiento del Lenguaje Natural y la Comprensión (NLP) del Lenguaje Natural (NLU), e incluso disciplinas como la lingüística computacional.
- La IA generativa en el contexto del texto significa generación de texto, traducción, reconocimiento de entidades, sistemas de diálogo, detección de sentimientos y un conjunto casi ilimitado de posibilidades.
- El NLP se ha visto impulsado con los últimos avances en aprendizaje profundo, enormes corpus de texto (Internet) y potentes sistemas computacionales (GPU, TPU, etc.)



2. Embeddings

La forma inteligente de asignar información discreta a formato numérico

- En el aprendizaje profundo, un **embedding** es una forma de representar un **objeto discreto**, como una palabra o una imagen, **en un espacio continuo de alta dimensión (un vector de números reales)**.
- Esto permite que el modelo aprenda una representación más compacta y útil de los datos, en lugar de tratarlos simplemente como un conjunto discreto y desordenado de símbolos.
- **Codificar los elementos discretos**, por ejemplo, las palabras, en solo índices o codificación fija, no es una buena idea. Como índices, el NN podría interpretar el índice como una magnitud, y con la codificación one-hot generamos un vocabulario grande donde la longitud del vector es la longitud del vocabulario, y una palabra es todo 0 excepto un 1.
- Los embeddings se utilizan a menudo en tareas de procesamiento de lenguaje natural, pero también se pueden utilizar, por ejemplo, en tareas de visión artificial, como la clasificación de imágenes, donde se pueden aprender de las imágenes y luego usarse para comparar la similitud entre diferentes imágenes.



2. Embeddings

One-hot encoding

- **One-hot encoding** es un proceso de conversión de variables de datos de carácter categórico para que puedan proporcionarse a los algoritmos de aprendizaje automático para mejorar las predicciones. En el **caso de las palabras**, la codificación one-hot implica representar cada palabra como vector de 'ceros', excepto un único 'uno'.
- Pongamos un ejemplo: si tenemos tres palabras 'cat', 'dog', 'mouse'. Podríamos representar estas palabras usando la codificación one-hot de la siguiente manera:

cat: [1, 0, 0]
dog: [0, 1, 0]
mouse: [0, 0, 1]

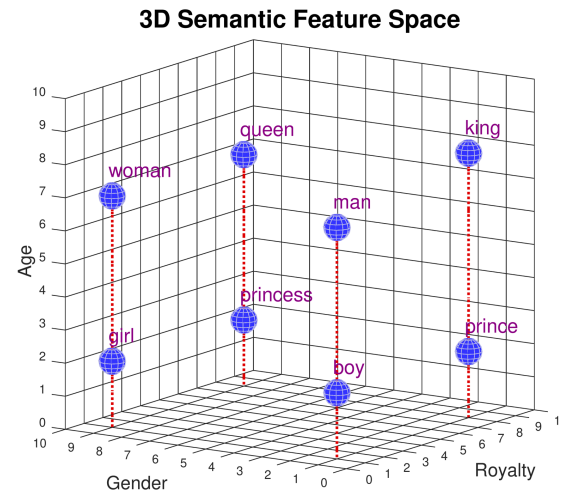
- Cada **palabra está representada por un vector de longitud igual al tamaño del vocabulario** (las palabras únicas en nuestros datos). El vector está formado enteramente por ceros, excepto por un solo uno en el índice correspondiente a esa palabra en el vocabulario.



2. Embeddings

Word embeddings

- Es mucho mejor algún tipo de representación vectorial donde codificar de alguna manera las relaciones de los diferentes elementos discretos o items. En el contexto de las **palabras**, es conveniente que **cada dimensión del vector sea una característica semántica. Esto podría darnos posibilidades muy interesantes.**
- Creando **embeddings como vectores con dimensiones de significado semántico**, se admiten operaciones algebraicas entre palabras que pueden revelar propiedades y patrones lingüísticos. Esto proporciona un marco matemático para razonar sobre la semántica del lenguaje.
- Podemos seleccionar estas características. Por ejemplo:



Word Coordinates			
	Gender	Age	Royalty
man	[1,	7,	1]
woman	[9,	7,	1]
boy	[1,	2,	1]
girl	[9,	2,	1]
king	[1,	8,	8]
queen	[9,	7,	8]
prince	[1,	2,	8]
princess	[9,	2,	8]



2. Embeddings

Word embeddings

- Sin embargo, como sabemos, una buena propiedad deseada de NN **es dejar que ellas decidan las características, eliminando el factor humano en esta selección.**
- Por otro lado, es bien sabido que el número de dimensiones, o rasgos semánticos, debe ser del orden de centenares.
- Por lo tanto, **usaremos una NN para encontrar estas características**, y lo haremos usando un **corpus grande**.
- De alguna manera, estos métodos tienen en cuenta las co-ocurrencias de palabras en oraciones en corpus grandes. Cuando procesan el corpus completo, **las palabras semánticas similares se asignan a regiones similares del hiperespacio de las características semánticas.**
- Por lo tanto, finalmente, las palabras con propiedades semánticas similares se asignarán a áreas similares de los hiperespacios de embeddings de palabras.
- La hipótesis fundamental que impulsa estos algoritmos es la **hipótesis distribucional**, que establece que las palabras que aparecen en contextos similares tienden a tener significados similares. Por ejemplo, si "apple" y "banana" aparecen a menudo en contextos relacionados con la fruta, la alimentación y la nutrición, el algoritmo aprende a asociarlos entre sí.
- **Al principio, a cada palabra se le asigna un vector aleatorio.** A lo largo del entrenamiento, estos vectores se ajustan gradualmente en función del contexto de la palabra.
- El propósito es posicionar palabras con significados o usos similares cerca unas de otras en el espacio de alta dimensión resultante, lo que permite a los vectores capturar relaciones semánticas entre palabras. Por ejemplo, los sinónimos se ubicarían muy juntos en el espacio vectorial, al igual que las palabras que a menudo se usan en un contexto similar, incluso si sus significados son diferentes.



2. Embeddings

Word embeddings

- La forma más común de crear embeddings es con una red neuronal. Esta red asigna cada palabra a un vector de números reales. Puede ser una red individual o como una capa en un proceso de más ambicioso sobre el corpus.
- **Algunas propiedades clave de los embeddings:**
 - Cada palabra está representada por un vector denso de tamaño fijo, ejemplo 300 dimensiones. Esto codifica el significado semántico de la palabra en los valores dentro del vector.
 - El embedding se aprende del contexto lingüístico de las palabras en los datos de entrenamiento. Las palabras utilizadas en contextos similares obtienen embeddings similares.
 - Los embeddings actúan como entradas de características para otras capas de la red neuronal (u otras redes). Los embeddings capturan el significado semántico y las relaciones entre las palabras.
 - Los embeddings son de baja dimensión en comparación con el tamaño del vocabulario. Esto añade eficiencia y generalización.
 - La principal ventaja de los embeddings es que proporcionan representaciones significativas de palabras que codifican relaciones semánticas. Esto permite a las redes neuronales generalizar patrones a través de palabras similares en función de sus embeddings. En general, los embeddings agregan más potencia expresiva en comparación con el uso de vectores codificados de manera fija.



2. Embeddings

Word embeddings

- Una vez que haya transformado sus palabras, oraciones o documentos en estas representaciones vectoriales numéricas, puede introducir estos vectores en varios algoritmos de aprendizaje automático o modelos de aprendizaje profundo (como redes neuronales convolucionales, redes neuronales recurrentes, transformers, etc.).
- Estos modelos pueden ayudar a resolver diferentes tareas de NLP como la clasificación de textos, el análisis de sentimientos, la traducción automática y la extracción de información, entre otras.



2. Embeddings

Algunas posibilidades en su obtención

- **Word2Vec:** Entrena una red neuronal superficial para reconstruir el contexto lingüístico de las palabras y, en el proceso, aprende vectores densos que tienen significados semánticos
- **GloVe** (Global Vectors for Word Representation): GloVe es similar a Word2Vec en el sentido de que también aprende vectores densos de palabras a través del entrenamiento, pero su enfoque es ligeramente diferente. Se entrena en las estadísticas globales agregadas de co-ocurrencia palabra-palabra de un corpus, con el objetivo de capturar información estadística global de palabras.
- **FastText:** Desarrollado por Facebook, FastText mejora Word2Vec al tratar cada palabra como compuesta de n-gramas de caracteres. Por lo tanto, en lugar de generar una incrustación de palabras para cada palabra, genera incrustaciones para cada n-grama de caracteres. Esto es particularmente útil para idiomas donde el significado de las palabras puede verse muy alterado por la adición de prefijos o sufijos.
- **BERT** (Bidirectional Encoder Representations from Transformers): BERT es un enfoque más reciente que utiliza una arquitectura de transformer para crear embeddings de palabras "contextuales". A diferencia de Word2Vec y GloVe, los embeddings de BERT cambian en función del contexto en el que se utiliza una palabra, lo que le permite capturar fenómenos lingüísticos complejos como la polisemia (donde una palabra puede tener múltiples significados).

Todos estos métodos sirven para transformar palabras en vectores numéricos, pero lo hacen de diferentes maneras y con diferentes capacidades. La elección de cuál utilizar a menudo depende de la tarea específica en cuestión.



2. Embeddings

Más información de cómo funcionan los métodos de embedding

- Consideremos una palabra como "apple". Después del entrenamiento, utilizando un método de generación de embeddings, se generará una representación vectorial para "apple". Esta sería una lista de 300 números (por ejemplo), como [0.23, -0.19, 0.87, ..., -0.56]. Este sería el embedding de "apple". Del mismo modo, todas las demás palabras de nuestro vocabulario tendrán su representación vectorial única.
- Estos métodos, a grandes rasgos, comienzan dando valores aleatorios a los vectores de palabras, y **la NN de alguna manera cambia estos valores al explorar el corpus al hacer que los vectores de palabras sean "más cercanos" en función de su co-ocurrencia en las oraciones.**
- La magia es que **estos vectores capturan similitudes semánticas y sintácticas**. Por lo tanto, palabras como "apple" y "pear" (ambas frutas) estarían más cerca en el espacio vectorial que palabras como "apple" y "car". Además, estos vectores pueden capturar analogías: la diferencia vectorial entre "king" y "man" podría estar cerca de la diferencia vectorial entre "queen" y "woman", capturando la relación de género entre estas palabras.
- Este proceso no está supervisado en absoluto: **el modelo aprende de los propios datos de texto sin procesar, sin necesidad de etiquetas o categorías explícitas**. Esto le permite capturar patrones semánticos y sintácticos en el lenguaje, como palabras que están relacionadas semánticamente que están más cerca en el espacio de incrustación.



2. Embeddings

Tokenization

La **tokenización** de subpalabras implica dividir las palabras en unidades más pequeñas (tokens). Por ejemplo, la palabra "unhappiness" podría dividirse en "un", "happy", y "ness".

Esto permite que estos modelos manejen palabras que no se han visto durante el entrenamiento, ya que pueden construir representaciones para estas nuevas palabras a partir de las subpalabras que han visto.

Podemos pensar en la tokenización como un proceso previo necesario antes de llevar a cabo los embeddings.



3. Recurrent Neural Networks

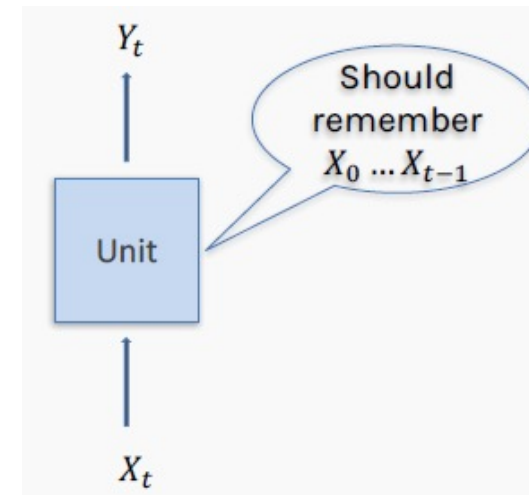
Memoria y secuencias

Las **Recurrent Neural Networks (RNN)** son especialmente buenas para trabajar con **secuencias** de información utilizando un estado oculto (una especie de memoria interna)

Aunque sus aplicaciones se centran principalmente en el NLP, es posible su uso con otra información donde el análisis de la secuencias de los datos o datos temporales es vital.

Hasta ahora, las NN que hemos visto son feed-forward o CNN, donde la entrada tiene un tamaño fijo y sólo una salida (para una entrada obtenemos finalmente una respuesta en la capa de salida).

En muchos problemas de ML, es interesante ser más flexibles, con arquitecturas que pueden tener una longitud variable en sus entradas y/o salidas.

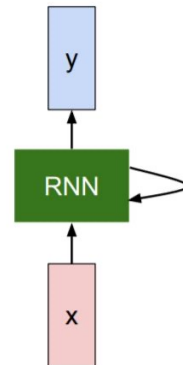




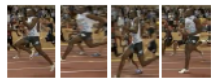


3. Recurrent Neural Networks

Memoria y secuencias

- Las RNN puede trabajar con información secuencial y series temporales.
- Guardan un estado oculto (o estado interno) que puede considerarse como un recuerdo de lo que la NN ha visto hasta el momento.
- Aplican una fórmula recurrente sobre la secuencia de entrada, por lo que, en cada paso de la secuencia, el proceso depende de la entrada x en este momento y del estado anterior h .
- Cada paso de la secuencia se denomina periodo de tiempo.



Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	\emptyset	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCGTGTGAGGAAGTAG	→	AGCCCCGTGTGAGGAAGTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger .

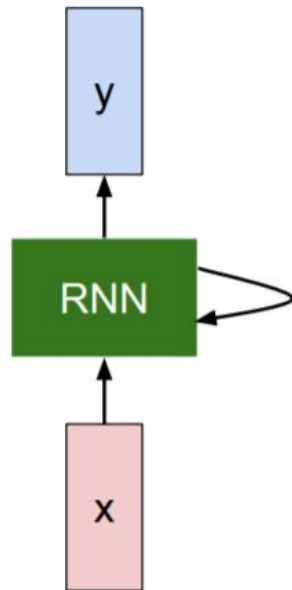


3. Recurrent Neural Networks

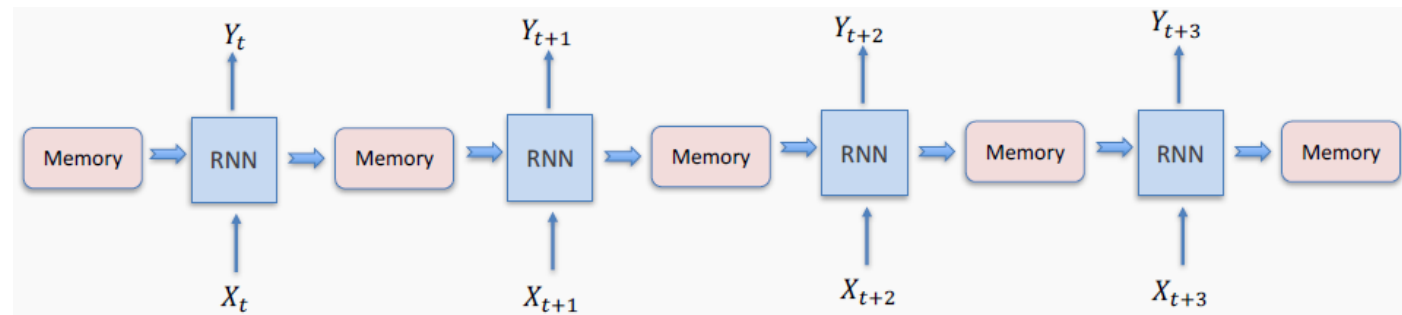
Memoria y secuencias

- RNN: Toma una secuencia de valores de entrada y , de forma recurrente, aplica una transformación a cada valor de entrada y al estado oculto de la red en este momento. Consiguiendo:
- **Un nuevo estado interno**
- **(Opcional) Un nuevo valor de salida**

$$h_t = f_W(h_{t-1}, x_t)$$



- h_t (vector) Estado oculto en el tiempo t . Su tamaño es un hiperparámetro.
- f_W (vector) transformación no lineal con W o pesos de la RNN
- x_t (vector) Entrada en el tiempo t

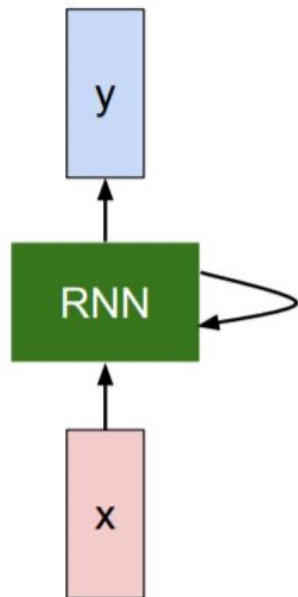




3. Recurrent Neural Networks

Memoria y secuencias

- RNN: Toma una secuencia de valores de entrada y , de forma recurrente, aplica una transformación a cada valor de entrada y al estado oculto de la red en este momento. Consiguiendo:
- **Un nuevo estado interno**
- **(Opcional) Un nuevo valor de salida**



$$h_t = f_W(h_{t-1}, x_t)$$

Ecuaciones para una RNN vainilla (una RNN simple):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

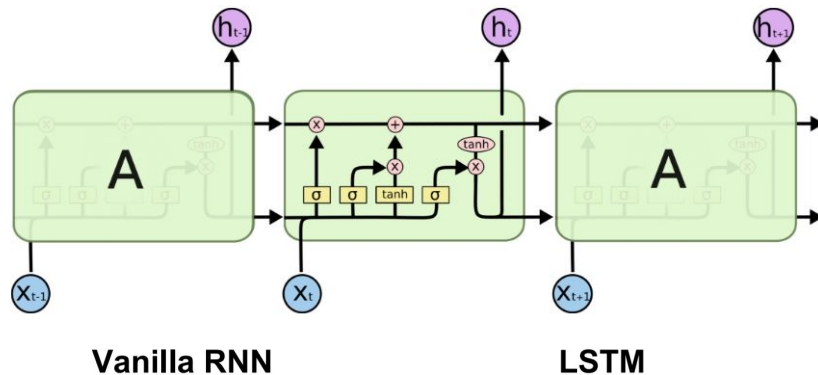
W_{hh} , W_{xh} , W_{hy} matrices de parámetros (pesos)



3. Recurrent Neural Networks

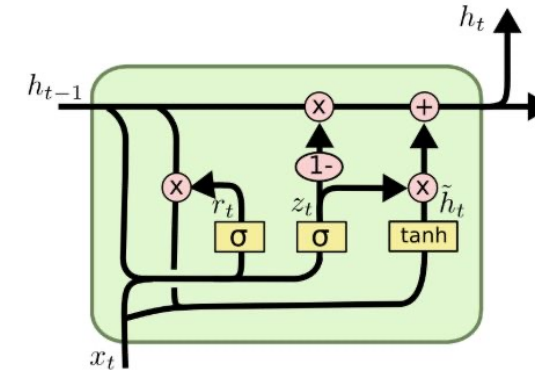
Memoria y secuencias

- Vanilla RNN tiene problemas con los gradientes (gradientes que se desvanecen y explotan) y con el fin de aprender relaciones con elementos con pasos de tiempo más distantes
- LSTM (Long Short-Term Memory) y GRU (gated recurrent unit)



$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$



4. Transformers

Más allá de RNN

Las RNN tienen varias limitaciones:

- **Vanishing/Exploding gradients:** Durante la retropropagación, los gradientes en las RNN pueden desaparecer (volverse muy pequeñas) o explotar (volverse muy grandes), lo que dificulta su entrenamiento efectivo en secuencias largas. Esto significa que las RNN pueden tener dificultades para mantener el contexto en secuencias más largas y, por lo tanto, tienen problemas para comprender las dependencias a largo plazo de los datos.
- **Cálculo secuencial:** Debido a la forma en que se diseñan las RNN, cada paso de la secuencia debe calcularse antes de pasar al siguiente, lo que significa que las RNN no se pueden paralelizar fácilmente. Esto conduce a tiempos de entrenamiento más largos.



4. Transformers

Más allá de RNN

Los **transformers** no dependen de la recurrencia; En su lugar, utilizan un mecanismo llamado "atención" para sopesar la importancia de diferentes palabras en la entrada al producir la salida.

Estas son algunas de las ventajas de los transformers sobre las RNN:

- **Control de dependencias a largo plazo:** Debido a que los Transformers usan la atención, pueden relacionar directamente palabras distantes en la entrada, lo que les permite manejar secuencias más largas y mantener el contexto de manera más efectiva que las RNN.
- **Cómputo paralelo:** Dado que los transformadores no tienen que procesar la secuencia en orden, se pueden paralelizar durante el entrenamiento, lo que lleva a tiempos de entrenamiento más rápidos.
- **Interpretabilidad:** El mecanismo de atención de Transformers puede proporcionar información sobre cómo el modelo está tomando decisiones al mostrar a qué partes de la entrada está prestando atención.

Si bien las RNN son útiles para los datos de secuencia y han tenido éxito en muchas tareas, los transformers son más efectivos en el manejo de secuencias largas y dependencias a largo plazo, y se pueden entrenar de manera más eficiente. Esto hace que transformers sea especialmente potente para tareas como la traducción de idiomas, la generación de texto y otras tareas complejas de NLP en las que el contexto y las dependencias a largo plazo son importantes.



4. Transformers

Una idea básica

La arquitectura transformer, introducida por Vaswani et al. en su artículo de 2017, 'Attention is All You Need', supuso una revolución en las tareas de procesamiento del lenguaje natural.

El transformer abordó estos problemas de la siguiente manera:

- Comienza tomando una secuencia de palabras de entrada.
- Cada palabra de la secuencia se convierte en un token (tokenización).
- A continuación, estos tokens se pasan a través de una capa de embedding, transformando cada token en un vector denso que encapsula su significado semántico aislado. En esta etapa, la palabra "perro" tendría el mismo embedding en cualquier oración que aparezca.
- Para conservar la información de orden de palabras perdida en el paso anterior, se agrega una codificación posicional a los embeddings. Esta codificación agrega información sobre la posición relativa o absoluta de las palabras en la secuencia. La belleza de esta adición es que aún permite que los cálculos se realicen en paralelo en todas las palabras, a diferencia de las RNN.



4. Transformers

Una idea básica

- A continuación, se emplea el mecanismo de **autoatención**, en el que el **embedding** de cada palabra se actualiza en función de las otras palabras de la frase. La influencia de otras palabras se pondera por su relevancia para la palabra actual, de ahí la palabra 'atención'. Ahora, "perro" puede tener diferentes embeddings según el contexto en el que aparece, ya que su codificación ahora refleja la influencia de las palabras vecinas.
- Múltiples cabezas de autoatención (multi-head attention) se ejecutan de forma independiente para capturar diferentes tipos de relaciones entre las palabras en la secuencia. Cada cabeza da una cierta cantidad de "atención" a otras palabras al codificar una palabra en particular. Todas las salidas de los cabezales de atención se combinan y procesan a través del resto de capas del transformer.



4. Transformers

Una idea básica

- Los pasos anteriores forman un 'bloque codificador' (**encoder block**), y varios de estos bloques se apilan para formar la parte codificadora del transformer. El propósito del codificador es comprender la oración de entrada y representarla de una manera que el decodificador pueda usar.
- A continuación, el decodificador toma esta entrada codificada y, utilizando un mecanismo similar de autoatención y codificaciones, genera la secuencia de salida paso a paso.
- Sin embargo, a diferencia del codificador, el decodificador es autorregresivo, lo que significa que genera una palabra a la vez y utiliza sus salidas anteriores como entradas adicionales para producir palabras posteriores. Este mecanismo permite que el transformer genere secuencias de salida como traducciones, resúmenes o cualquier forma de generación de texto.



5. Similitud semántica

Saber qué tan similares son dos conceptos

- La **similitud semántica** es una medida del grado en que dos piezas de texto están relacionadas en términos de significado. Por ejemplo, "gato" y "gatito" o "perro" y "cachorro" son semánticamente similares porque ambos representan un tipo de animal y tienen una relación padre-hijo.
- Cuando se trata de embeddings de palabras u oraciones, la similitud semántica se calcula normalmente utilizando la similitud de coseno.
- Los embeddings de palabras son vectores multidimensionales, donde cada dimensión representa alguna característica semántica de una palabra.
- Si los embeddings han sido bien generados con un corpus representativo, entonces las palabras similares deberían tener vectores similares. Esta similitud se puede calcular tomando el coseno del ángulo entre estos vectores.



5. Similitud semántica

Saber qué tan similares son dos conceptos

La similitud del coseno se calcula de la siguiente manera:

$$\text{cosine_similarity}(A, B) = \text{dot_product}(A, B) / (\|A\| * \|B\|),$$

Dónde:

- A y B son nuestros vectores de palabras,
- $\text{dot_product}(A, B)$ es el producto escalar de A y B,
- $\|A\|$ y $\|B\|$ son las magnitudes (o longitudes) de los vectores A y B.
- La similitud del coseno producirá una métrica que te dará el coseno del ángulo entre los dos vectores. Este será un número entre -1 y 1, donde 1 indica una similitud muy alta y -1 indica que los vectores son diametralmente opuestos.



5. Similitud semántica

Saber qué tan similares son dos conceptos

El embedding de una **oración** se puede calcular a partir de embeddings de palabras (para calcular similitud semántica entre oraciones) promediando los embeddings de las palabras en la oración. Esto significa que el embedding de la oración capturará el significado de toda la oración, en lugar de solo el significado de las palabras individuales.

A continuación se ofrece una explicación más detallada de cómo se calcula embeddings de frases por promedio:

- **Word embeddings:** El primer paso es crear embeddings de palabras para las palabras de la oración.
- **Averaging:** Una vez que se han creado los embeddings de las palabras, se promedian para crear el embedding de oraciones.

A continuación, se utiliza el embedding de la frase para representar el significado de la frase.

Esto puede ser útil para RAG (Retrieval-Augmented Generation) donde la información externa se divide en chunks (teniendo en cuenta los límites de token del transformer) para inyectar información en el prompt.

Existen otras formas de abordar el cálculo de embeddings de frases o documentos (usar RNNs, por ejemplo, y quedarnos con el estado interior como embedding final, usar los propios mecanismos atencionales del transformer etc.)



6. Retrieval-Augmented Generation

RAG

Retrieval-Augmented Generation (RAG) es una técnica que combina las fortalezas de la recuperación y la generación para crear respuestas más informativas y completas. Los sistemas RAG primero recuperan información relevante de una base de conocimientos o un conjunto de datos externo y, a continuación, utilizan un modelo de lenguaje para generar una respuesta adaptada a la solicitud específica.

RAG se utiliza en una variedad de contextos, que incluyen:

- **Responder a las preguntas:** El RAG se puede utilizar para responder preguntas que requieren información objetiva. Por ejemplo, si un usuario pregunta "¿Cuál es la capital de Francia?", un sistema RAG podría recuperar información de una base de conocimientos para responder a la pregunta.
- **Texto resumido:** RAG se puede utilizar para resumir texto recuperando información relevante del texto y luego generando un resumen que capture los puntos principales del texto.
- **Generación de texto creativo:** RAG se puede utilizar para generar texto creativo, como poemas, historias o código. Por ejemplo, si un usuario pregunta "Escríbeme un poema sobre un gato", un sistema RAG podría recuperar información sobre gatos de una base de conocimientos y luego generar un poema sobre gatos.



6. Retrieval-Augmented Generation

RAG

El RAG se utiliza porque puede producir respuestas más informativas y completas que la recuperación o la generación por sí solas (utilizando únicamente el conocimiento paramétrico del modelo).

Retrieval puede proporcionar información objetiva, pero no puede generar texto creativo. **Generation** puede generar texto creativo, pero puede ser inexacto o sesgado. RAG combina las fortalezas de la recuperación y la generación para producir respuestas más precisas, completas e informativas.

Estos son algunos de los beneficios de usar RAG:

- **Resolución más informativa y completa:** El RAG puede producir respuestas más informativas y completas que la recuperación o la generación por sí solas.
- **Respuestas más creativas:** RAG puede generar texto creativo que se adapte a la indicación específica.
- **Respuestas más precisas:** El RAG puede ser más preciso que la generación sola, ya que puede recuperar información objetiva de una base de conocimientos.
- **Respuestas menos sesgadas:** El RAG puede estar menos sesgado que la generación sola, ya que puede recuperar información de una variedad de fuentes.



6. Retrieval-Augmented Generation

RAG

Uno de los pasos involucrados en RAG es el preprocesamiento de la información externa utilizando **chunks**.

Chunks son pequeñas piezas de información que se extraen de la información externa. Por lo general, tienen un tamaño limitado. Esto se debe a que, por lo general, los modelos de lenguaje no pueden procesar grandes cantidades de texto a la vez (límite de tokens).

El límite de tokens es el número máximo de tokens que un modelo de lenguaje puede procesar a la vez. Normalmente, este límite lo establece el propio modelo de lenguaje. Por ejemplo, el modelo de lenguaje GPT-3 tiene un límite de tokens de 1024 tokens, GPT-3.5 4K, GPT-4 hasta 32K (8K inicialmente), Claude de Anthropic 200K, etc.

La razón por la que los fragmentos deben ser compatibles con los límites de tokens es porque los modelos de lenguaje no pueden procesar fragmentos que sean mayores que el límite de tokens. Si un fragmento es mayor que el límite de tokens, el modelo de lenguaje no podrá procesarlo y devolverá un error.



6. Retrieval-Augmented Generation

RAG

Estos son algunos de los beneficios de usar chunks:

- **Los chunks son más fáciles de procesar:** Normalmente, los modelos de lenguaje son capaces de procesar fragmentos más fácilmente que grandes cantidades de texto. Esto se debe a que los trozos son más pequeños y manejables.
- **Los chunks se pueden reutilizar:** Chunks se puede reutilizar en diferentes respuestas. Esto puede ahorrar tiempo y esfuerzo, ya que el modelo de lenguaje no necesita procesar la misma información varias veces.
- **Chunks se puede indexar:** Chunks se puede indexar, lo que facilita la búsqueda de fragmentos relevantes para un mensaje determinado. Esto puede mejorar la precisión del sistema RAG.



6. Retrieval-Augmented Generation

RAG

En RAG, la similitud semántica se utiliza para encontrar fragmentos de información relevantes para un mensaje específico. Esto se hace comparando el mensaje con los fragmentos de información y encontrando los fragmentos que son más similares semánticamente al mensaje.

Una vez que se han encontrado los fragmentos de información más similares semánticamente, se pueden usar para generar una respuesta a la solicitud. La respuesta puede ser generada por un modelo de lenguaje, como GPT. El modelo de lenguaje utilizará los fragmentos de información para crear una respuesta que sea a la vez informativa y completa.

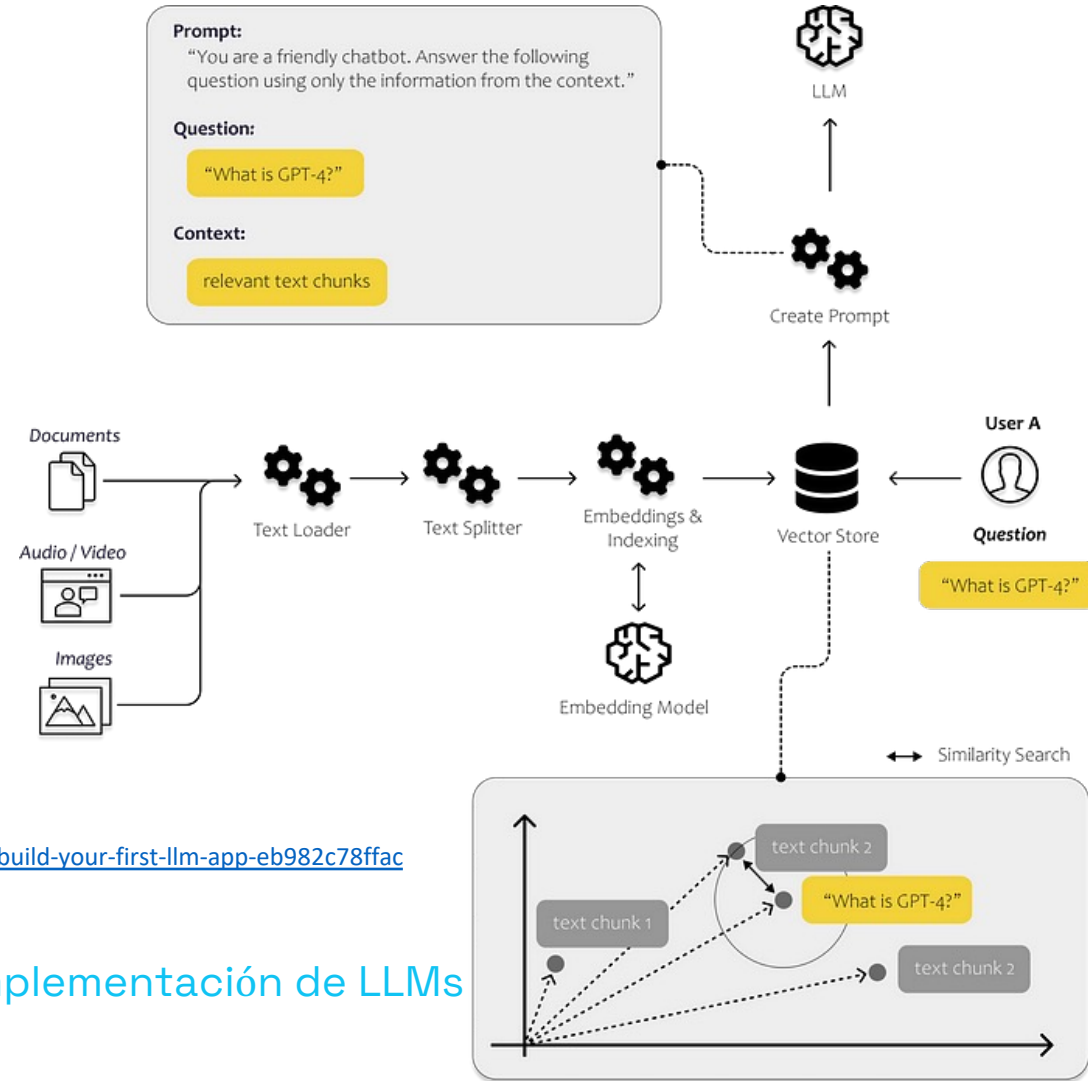
Estos son algunos de los beneficios de usar la similitud semántica en RAG:

- **Respuestas más relevantes:** La similitud semántica puede ayudar a encontrar fragmentos de información más relevantes para un mensaje específico. Esto puede mejorar la precisión del sistema RAG.
- **Respuestas más informativas:** La similitud semántica puede ayudar a encontrar fragmentos de información que sean más informativos. Esto puede mejorar la calidad de las respuestas generadas por el sistema RAG.
- **Respuestas más completas:** La similitud semántica puede ayudar a encontrar fragmentos de información que sean más completos. Esto puede mejorar la exhaustividad de las respuestas generadas por el sistema RAG.



6. Retrieval-Augmented Generation

RAG



<https://towardsdatascience.com/all-you-need-to-know-to-build-your-first-llm-app-eb982c78ffac>



6. Retrieval-Augmented Generation

RAG versus model fine tuning

Model fine-tuning es una técnica que mejora el rendimiento de un modelo de lenguaje entrenándolo en una tarea específica. Esto se hace proporcionando al modelo de lenguaje un conjunto de datos de ejemplos de la tarea en la que se está entrenando (enfoque supervisado).

Normalmente, el fine-tuning no es interesante para inyectar nuevo conocimiento, sino para alinear más el modelo de lo que esperamos de él cuando le suministramos las entradas.

RAG o fine-tuning:

- Si la aplicación requiere respuestas informativas y completas, entonces RAG puede ser una mejor opción.
- Sin embargo, si la aplicación requiere una alta precisión en una tarea específica, el ajuste fino del modelo puede ser una mejor opción.

Ambos enfoques también pueden combinarse.

