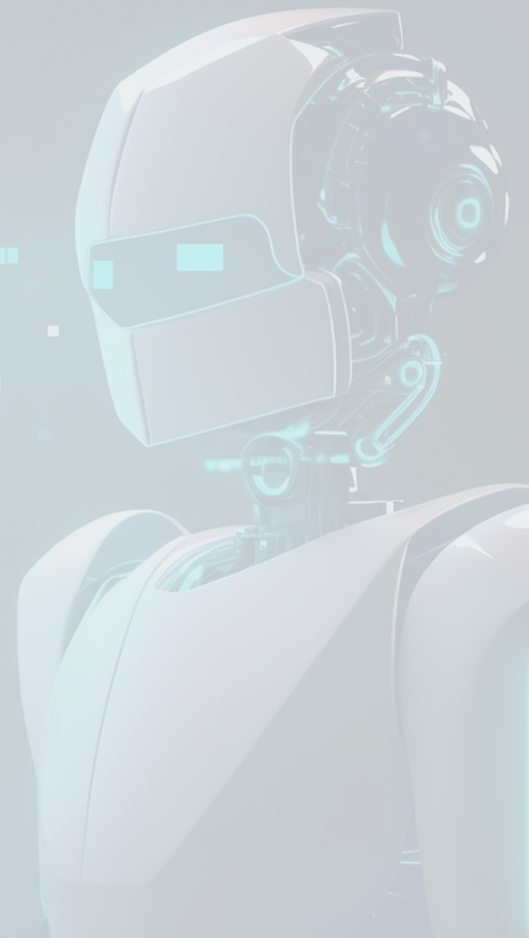




Introduction to Large Language Models and Agents

Ronald Cardenas Acosta, Ph.D.
Investigador en Procesamiento de Lenguaje Natural



Syllabus



Unidad 1	Conceptos Básicos
Unidad 2	Modelado de Lenguaje
Unidad 3	Entrenamiento y Fine-tuning
Unidad 4	Aplicaciones de LLM

Unidad 2: Modelado de Lenguaje

Modelos de Lenguaje

Word Embeddings

Redes Recurrentes

Transformers

Trabajo de Laboratorio 1



Information Theory, *the essentials*

Nocion de Entropia



- Entropia ~ “caos”, lo opuesto al orden, confusion
- Importado de fisica
 - La entropía no disminuye a menos que se aplique energía
- Medida de incertidumbre
 - Baja entropía -> baja incertidumbre
 - alta entropía -> alta incertidumbre
 - Pero mas alta la “sorpresa” (informacion) que podemos obtener de un experimento

Entropia



- Sea:
 - $p(X)$: distribución de probabilidad sobre la variable aleatoria X
 - χ : conjunto de posibles resultados (*espacio muestral*)

$$H(X) = - \sum_{x \in \chi} p(x) \log_2 p(x)$$

- Unidades: bits (log10: nats)
- Notacion: $H(X) = H(p)$

Perplexity



$$PPL(p) = 2^{H(p)}$$

- Una métrica más intuitiva que entropía
 - PPL=32 cuando hay 32 resultados equiprobables
 - PPL=2 para una moneda sin adulterar
 - ...
- Ampliamente usado en NLP, ejemplo
 - PPL = tamaño del vocabulario cuando todas las palabras son equiprobables
- Mientras más sesgada (biased) la distribución, mejor
 - Menos entropía, menos perplejidad

Entropia Conjunta y Condicional



- Dadas dos variables aleatorias X, Y con espacios muestrales χ, ψ
- Entropia Conjunta (Joint Entropy)

$$H(X, Y) = - \sum_{x \in \chi} \sum_{y \in \psi} p(x, y) \log p(x, y)$$

- Entropia Condicional (Conditional Entropy)

$$H(Y|X) = - \sum_{x \in \chi} \sum_{y \in \psi} p(x, y) \log p(y|x)$$

Entropía como “*codificación*”



- “ $H(p)$ es el número mínimo (en promedio) de bits necesarios para codificar un mensaje (texto, secuencia, senhal, ...)”
- Algoritmos de compresión
 - Más efectivos en data con patrones repetidos (fácilmente predecibles = baja entropía)

Entropía de un Lenguaje

- Digamos que generamos la siguiente letra usando

$$p(l_{n+1}|l_1, \dots, l_n)$$

Donde l_1, \dots, l_n es la secuencia de **todas las letras** escritas hasta ahora

- Sea l_1, \dots, l_n el *historial*, y H el conjunto de todos los historiales
- Entonces, la entropía se define como

$$-\sum_{h \in H} \sum_{l \in \Lambda} p(l, h) \log p(l|h) \quad \text{impractico!}$$



Modelos Probabilísticos de Lenguaje

Modelos de Lenguaje: Motivacion



- Traducción automática

EN: "The cat eats fish"

ES: $P(\textit{el gato come pescado}) > P(\textit{el gato come pez})$

- Corrección gramatical

$P(\textit{el gato}) > P(\textit{el gtao})$

- Reconocimiento de voz

$P(\textit{vi una casa}) > P(\textit{ve oaac asa})$

Modelos de Lenguaje



- Evento u Observación: **unidad de lenguaje**
 - Letras, silabas, palabras, ...

- Objetivo:

Aproximar la probabilidad de una secuencia de unidades de lenguaje

$P(w_1, w_2, w_3, \dots, w_N)$ Probabilidad conjunta de una secuencia
(e.g. de palabras)

$P(w_i \mid w_1 \dots w_{i-1})$ Probabilidad condicional de la siguiente
unidad (*next-token prediction*)



Modelos de N-gramas

Modelos de Lenguaje



$W = \text{"me gusta correr"}$

$$P(\text{me,gusta,correr}) = P(\text{me}) * P(\text{gusta} \mid \text{me}) * P(\text{correr} \mid \text{me, gusta})$$

Parametros del modelo (distribuciones que tenemos que aproximar):

- $P(X)$
- $P(Y \mid X)$
- $P(Z \mid X, Y)$

X, Y, Z : pueden tomar cualquier palabra del vocabulario

Modelos de Lenguaje

Sea la probabilidad de una oración $\mathbf{W}=\langle w_1...w_n \rangle$

$$P(w_1, \dots w_M) = \prod_{i=1}^M P(w_i | w_1, \dots w_{i-1})$$

Regla de la cadena + Regla de Bayes

Problema

- Cada w_i usa *todos sus predecesores*
- Aun para n pequeno -> demasiados parametros!

Solucion

- Modelar $P(W)$ como una Cadena de Markov
- Cada w_i depende *solo de un número fijo de predecesores*

Modelos de Lenguaje de N-gramas



- Aproximación de Markov de Orden (**n-1**) => modelo n-grama

$$P(W) = \prod_i^M P(w_i | w_{i-n}, w_{i-n+1}, \dots, w_{i-1})$$

- w_i depende solo de n predecesores
- Para un vocabulario $|V|=60k$
 - $n=0$: modelo 0-gram, *uniforme*, $p(w) = 1/|V|$: 1 parametro
 - $n=1$: modelo 1-gram, unigrama, $p(w)$: 6×10^4 parametros
 - $n=2$: modelo 2-gram, bigrama, $p(w)$: 3.6×10^9 parametros
 - $n=3$: modelo 3-gram, trigram, $p(w)$: 2.16×10^{14} parametros

Modelo N-gramas: Observaciones



- Limitaciones
 - Cuanto mayor n , mejor, pero impractico para $n > 4$
 - Palabras puede tener dependencias muy separadas entre sí

“La vasija que deje en el cuarto piso se cayó”

- En practica
 - $n=3$ (modelo trigramas) -> suficientemente bueno

Modelo N-gramas: Como estimar parametros

Maximum Likelihood Estimate

Dado un vocabulario V de palabras, y un dataset D

$$p(w) = \frac{c(w)}{|V|}$$

$$p(w_2|w_1) = \frac{c(w_1, w_2)}{c(w_1)}$$

$$p(w_3|w_1, w_2) = \frac{c(w_1, w_2, w_3)}{c(w_1, w_2)}$$

...

- $|V|$: tamanho del vocabulario
- $c(W)$: número de veces n-grama W aparece en D

Modelo N-gramas: Como estimar parametros

Maximum Likelihood Estimate: Ejemplo

$$p(w_2|w_1) = \frac{c(w_1, w_2)}{c(w_1)}$$

D = <s> yo soy Juan </s>
<s> Juan yo soy </s>
<s> yo no salto </s>

$$p(\text{yo} \mid \text{<s>}) = 2/3$$

$$p(\text{soy} \mid \text{yo}) = 2/3$$

$$p(\text{Juan} \mid \text{soy}) = 1/2$$

$$p(\text{</s>} \mid \text{Juan}) = 1/2$$

...

$$p(\text{Juan} \mid \text{<s>}) = 1/3$$

$$p(\text{no} \mid \text{yo}) = 1/3$$

$$p(\text{</s>} \mid \text{soy}) = 1/2$$

$$p(\text{yo} \mid \text{Juan}) = 1/2$$

Evaluación de un modelo de lenguaje



- Todo modelo NLP puede ser evaluado de manera **intrínseca** y/o **extrínseca**
- **Evaluación intrínseca**
 - Modelo es evaluado en la tarea en la que fue entrenado
- **Evaluación extrínseca**
 - Modelo es evaluado en tareas en las que contribuye o interactúa (*downstream tasks*)
 - Caso: Modelado de Lenguaje
 - Downstream tasks: Machine translation, speech recognition, ...

Evaluación intrínseca de un modelo de lenguaje

- Tarea: aproximar distribución $P(W)$
- Criterio de evaluación
 - Cuán bien nuestro modelo aproxima la “distribución real”
- Métrica: **Perplexity**

$$PPL(p) = 2^{H(p)}$$

$$PPL(p) = 2^{-\sum_M p(W) \log p(W)}$$

$$PPL(p) = \sqrt[M]{\frac{1}{p(W)}}$$

$$PPL(p) = \sqrt[M]{\frac{1}{\prod_i^M p(w_i | w_1 \dots w_{i-1})}}$$

Minimizar PPL <>
maximizar probabilidad

Generando texto con Modelos de Lenguaje



- Selecciona o muestra una palabra a la vez, condicionada en las palabras ya seleccionadas

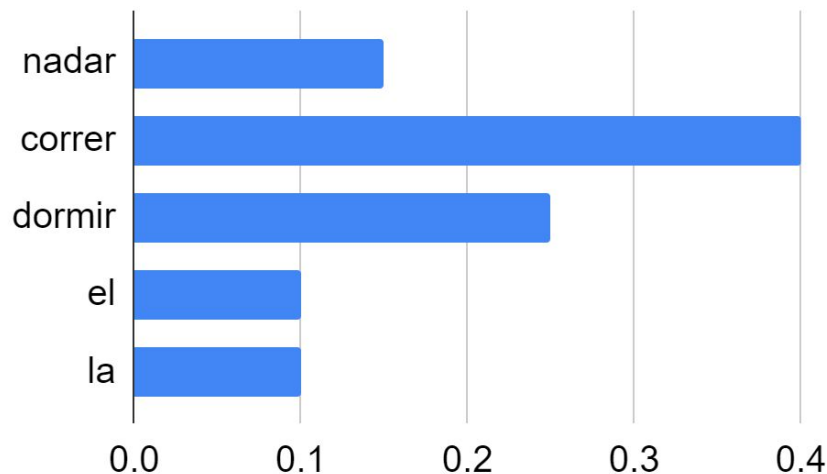
$$a_i \sim P(*|\underbrace{a_1, \dots, a_{i-1}})$$

Contexto o Historia

Generando texto con Modelos de Lenguaje

“Me gusta ____”

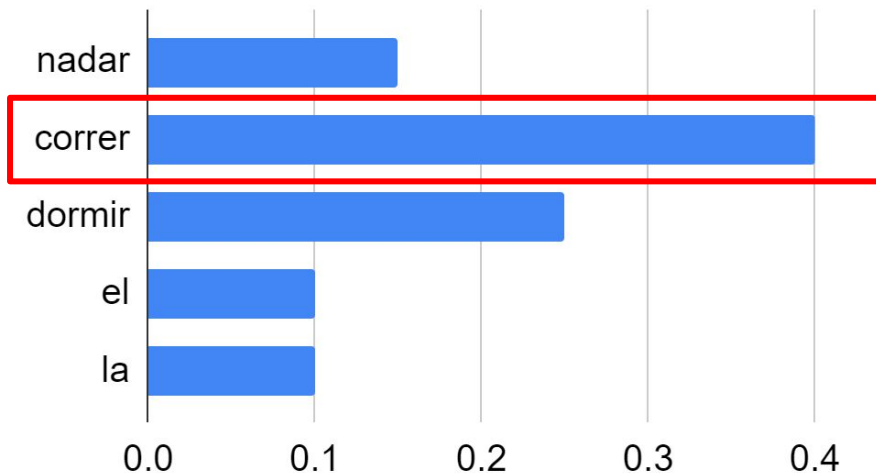
$P(* | \text{me,gusta}) :$



Generando texto con Modelos de Lenguaje

“Me gusta ____”

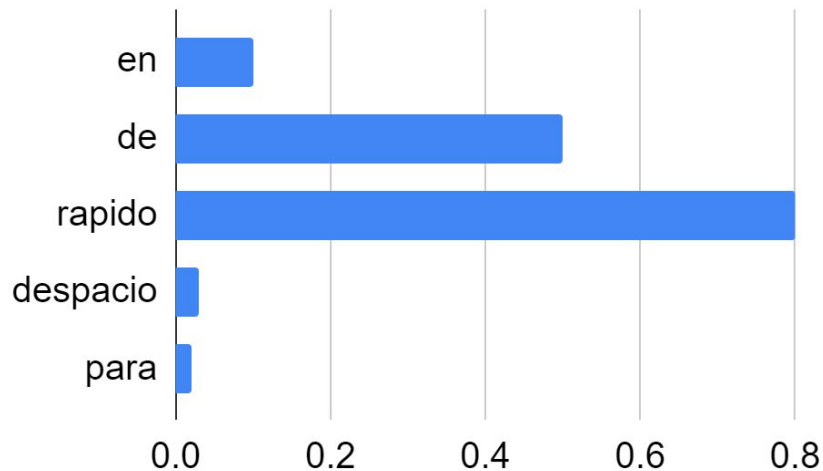
$P(* | \text{me,gusta}) :$



Generando texto con Modelos de Lenguaje

“Me gusta correr ____”

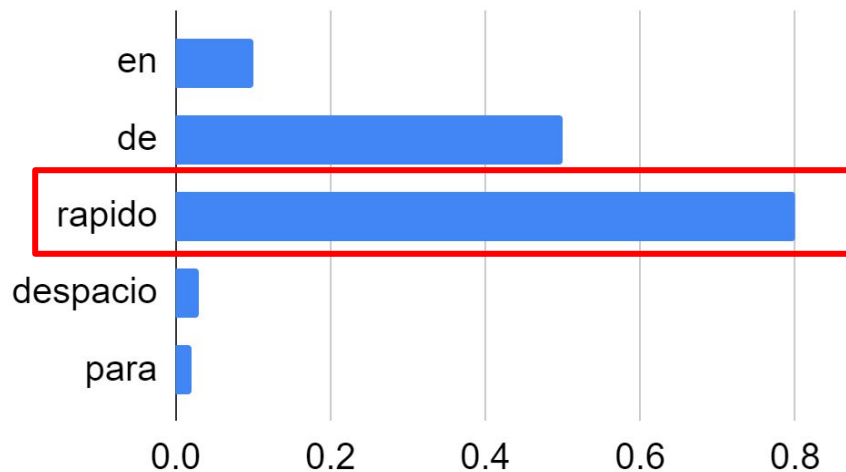
$P(* | \text{me,gusta,correr}) :$



Generando texto con Modelos de Lenguaje

“Me gusta correr ____”

$P(* | \text{me,gusta,correr}) :$





Representando Palabras: Word Embeddings

Word embeddings

“mesa”



$$E(\text{“mesa”}) = \begin{bmatrix} 0.1 \\ 0. \\ 0.9 \end{bmatrix} \left\{ \right.$$

Vector que capture

- El significado de la palabra -> ***Semántica***

$$||E(\text{“mesa”}) - E(\text{“silla”})|| < ||E(\text{“mesa”}) - E(\text{“auto”})||$$

- Su función gramatical -> ***Sintáctica***

$$||E(\text{“mesa”}) - E(\text{“silla”})|| < ||E(\text{“mesa”}) - E(\text{“escalar”})||$$

Word embeddings: uso en NLP

Cualquier modelo que resuelva una tarea

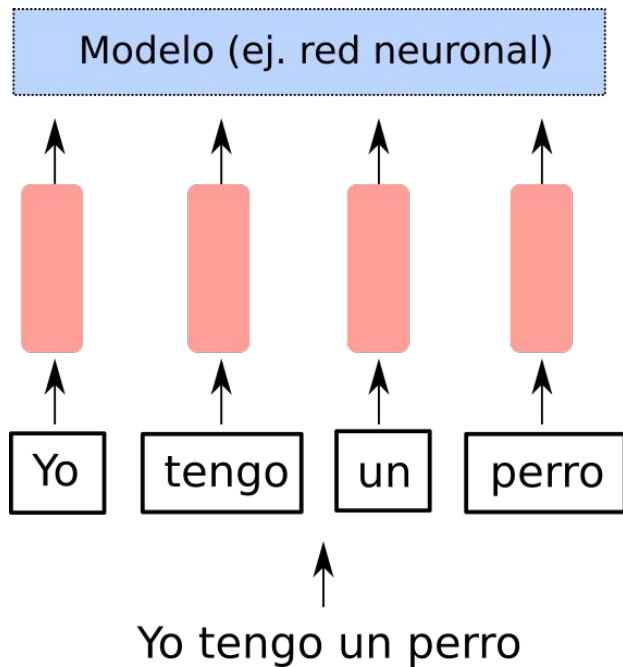
Representación de cada palabra
(input para el modelo)

Secuencia de tokens (*Tokenization*)

Input del usuario



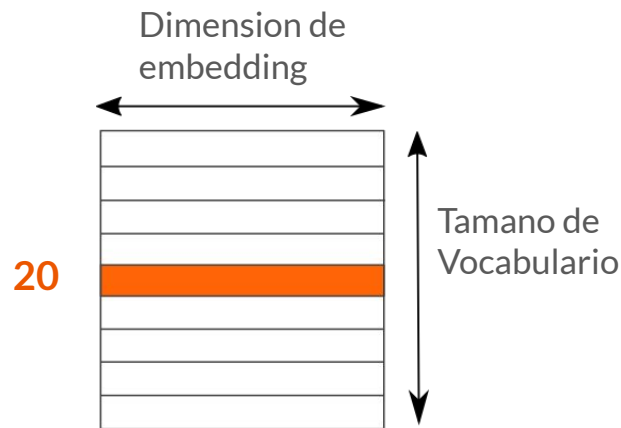
Texto (input)



Word embeddings: Look-up Table

- Vocabulario V -> conjunto de palabras permitidas
 - Si w no esta en V , se mapea al token $\langle UNK \rangle$
- Look-up Table
 - Estructura de datos que almacena los embeddings de todo w en V

Yo tengo un perro
20 1430 132 6



Contexto, palabras, y significado

(1) Un pequeño **tahualaco** se posa en un árbol.

Que significa **tahualaco**?

(2) Los **tahualacos** exhiben vistosas plumas.

{

El Tahualaco es un ave

(3) La velocidad de un **tahualaco** es impresionante.

Con el contexto podemos deducir el significado de una palabra!

Contexto, palabras, y significado

- (1) Un pequeño _____ se posa en un árbol.
- (2) Los _____ exhiben vistosas plumas.
- (3) La velocidad de un _____ es impresionante.

Qué otras palabras encajan en estos contextos?



	(1)	(2)	(3)
tahualaco	1	1	1
mesa	0	0	0
papagallo	1	1	0
lamborgini	0	0	1

Fila: propiedades contextuales

- palabra puede encajar -> 1
- Sino -> 0

Contexto, palabras, y significado

- (1) Un pequeño _____ se posa en un árbol.
- (2) Los _____ exhiben vistosas plumas.
- (3) La velocidad de un _____ es impresionante.

Qué otras palabras encajan en estos contextos?



Filas son similares → Significados son similares?

Hypothesis Distribucional

	(1)	(2)	(3)
tahualaco	1	1	1
mesa	0	0	0
papagallo	1	1	0
lamborgini	0	0	1

Como obtener Word Embeddings:

Semantica Distributiva

Hypothesis distribucional:

*“Palabras que co-ocurren frecuentemente en **contextos similares** tienen **significado similar**”*

Objetivo:

Codificar información sobre contextos dentro de representaciones de palabras

Como obtener Word Embeddings:

Semantica Distributiva

Hypothesis distribucional:

*“Palabras que co-ocurren frecuentemente en **contextos similares** tienen **significado similar**”*

Objetivo:

Codificar información sobre contextos dentro de representaciones de palabras

Word2Vec



Objetivo: *Codificar información sobre contextos dentro de representaciones de palabras*

Como: Entrenar vectores enseñándoles a predecir sus contextos

Word2Vec: idea principal

- Dado un corpus textual grande (>millones de palabras)
- Itera el texto con una ventana deslizante, una palabra a la vez
- Dada la palabra central, calcula las probabilidades de las palabras en su contexto (ventana)
- Ajusta los vectores para incrementar estas probabilidades

$$P(w_{t-2}|w_t) \quad P(w_{t-1}|w_t) \quad P(w_{t+1}|w_t) \quad P(w_{t+2}|w_t)$$

... Yo vi un ave volando en cielo ...

w_{t-2} w_{t-1} w_{t+1} w_{t+2}

palabras **palabra** palabras
contexto **central** contexto

Word2Vec: entrenamiento

- Objetivo: ajustar los parámetros que maximizan el log-likelihood de la data

Likelihood

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

- Loss a minimizar: **Negative Log-Likelihood**

Como calculamos P

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

Itera el
texto

con una
ventana

calcula la probabilidad del
contexto dado la
palabra central

Word2Vec: Calculando $P(w_{t+j}|w_t, \theta)$

- Para cada palabra w , definimos dos vectores
 - v_w : cuando w es palabra central
 - u_w : cuando w es palabra contextual
- Parametros θ : Conjunto de v, u para cada palabra en el vocabulario
- Una vez entrenados, descartamos u y usamos solo v como **embedding**

$$P(a|b) = \frac{\exp(u_a^T v_b)}{\sum_{w \in V} \exp(u_w^T v_b)}$$

Producto escalar: medida de similaridad entre u y v
Mayor valor \rightarrow mayor probabilidad

Normalizamos sobre todo el vocabulario para
obtener una distribución de probabilidad válida

Muestreo Negativo (Negative Sampling)

Normalizando sobre todo V

- Parámetros a actualizar en cada paso:
 v_b , u_w para todo w en $V = |V| + 1$
- Ineficiente: $|V|$ puede ser muy alto

$$P(a|b) = \frac{\exp(u_a^T v_b)}{\sum_{w \in V} \exp(u_w^T v_b)}$$

Normalizando sobre subconjunto de V (V')

- Parámetros a actualizar en cada paso:
 v_b , u_a , y u_w para todo w en V'
 $\rightarrow |V'| + 2$
- Viable: $|V'|$ usualmente < 20

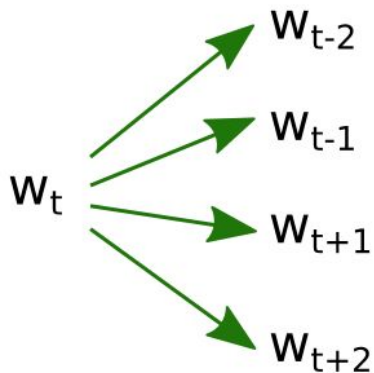
Loss : Negative Sampling

$$J_{t,j}(\theta) = -\log \sigma(u_j^T v_t) - \sum_{w \in \{w^1, \dots, w^k\}} \log \sigma(-u_w^T v_j)$$

Variantes de Word2Vec: Skip-gram y CBOW

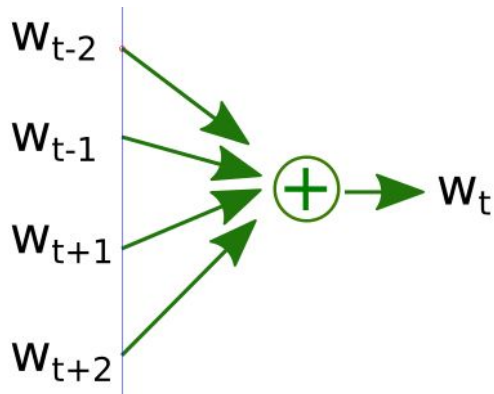
Skip-Gram

Dada la palabra **central**, predecir el **contexto** (uno a la vez)



CBOW

Dada **suma del contexto**, predecir la palabra **central**



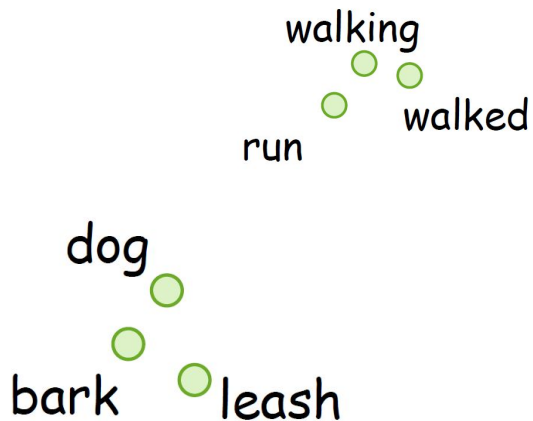
Word2Vec: Tips & Hyper-parametros

- **Variante:** Skip-gram con negative sampling
- **Número de muestras negativas ($|V'|$)**
 - Para datasets pequenos, 15 - 20
 - Para datasets grandes, 2-5
- **Dimensionalidad de los vectores:** 300, 100, 50
- **Tamaño de ventana deslizante:** 5 - 10

Word2Vec: Efecto del tamaño del contexto

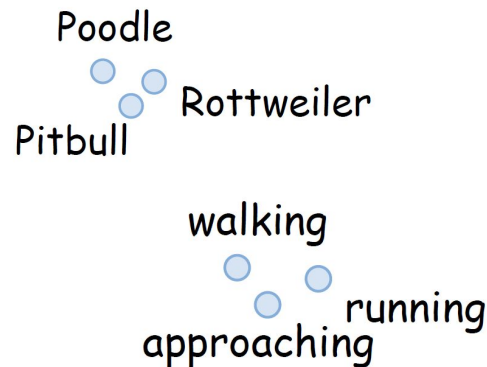
Ventana más larga = mayor contexto

Similaridades mas topicales



Ventana más pequeña = menor contexto

Similaridades mas gramaticas



Word Embeddings: Analogías

- Los vectores aprendidos pueden ser usados para resolver analogías con simples *operaciones algebraicas*

“king” - “man” + “woman” ~ “queen”

```
model.most_similar(positive=["king", "woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),  
 ('throne', 0.7664333581924438),  
 ('prince', 0.7592144012451172),  
 ('daughter', 0.7473883032798767),  
 ('elizabeth', 0.7460219860076904),  
 ('princess', 0.7424570322036743),  
 ('kingdom', 0.7337411642074585),  
 ('monarch', 0.721449077129364),  
 ('eldest', 0.7184862494468689),  
 ('widow', 0.7099430561065674)]
```

Word Embeddings: visualización



<https://projector.tensorflow.org/>



Modelos Neuronales de Lenguaje

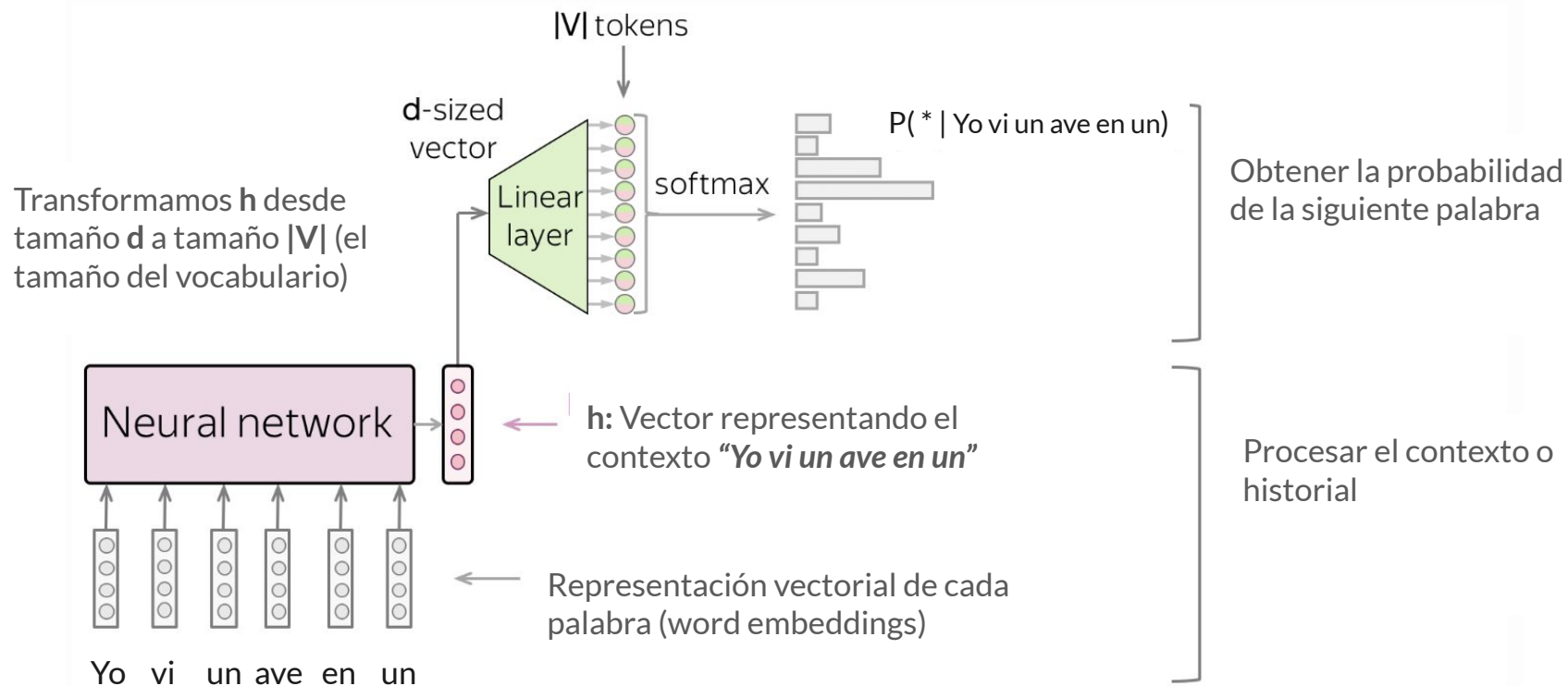
Idea general

Cross-Entropy

Redes Recurrentes

Transformer

Modelos Neurales de Lenguaje



Entrenamiento: Cross-Entropy

Yo vi un **ave** en un árbol <eos>

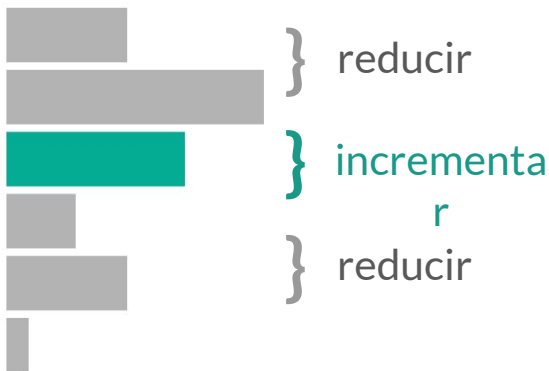
Predicción del
Modelo:
 $p(* | \text{Yo vi un})$



Objetivo:
 p^*



Loss:
 $J(\theta) = -\log p(\text{ave})$



$$J(\theta) = -\sum_t^T \sum_i^{|V|} p_i^* \cdot \log P(y_t = y_i | x)$$

Entrenamiento: Cross-Entropy



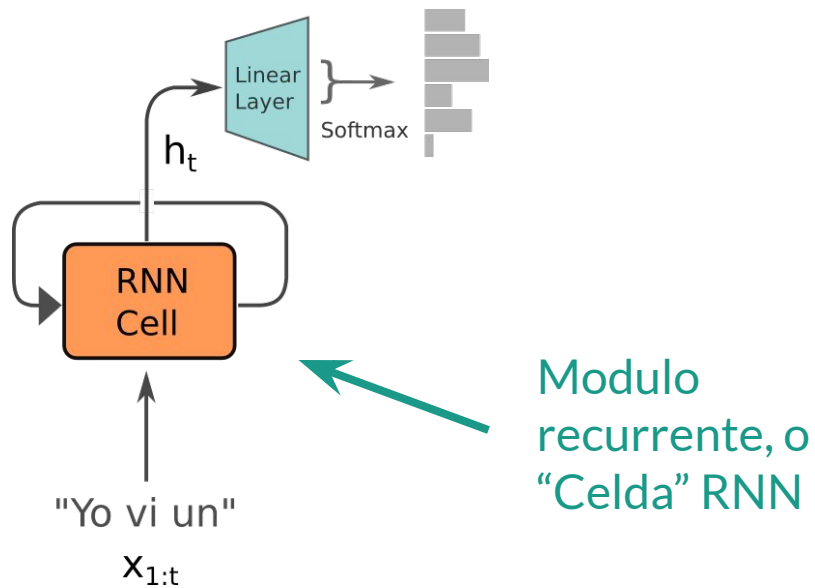
$$J(\theta) = -\frac{1}{T} \sum_t^T \sum_i^{|V|} p_i^* \cdot \log P(y_t = y_i | x)$$

- Modelo de clasificación de $|V|$ clases
- Opción de-facto en entrenamiento en la tarea de next-token prediction
- Generalización de Logistic Regression (2 clases)

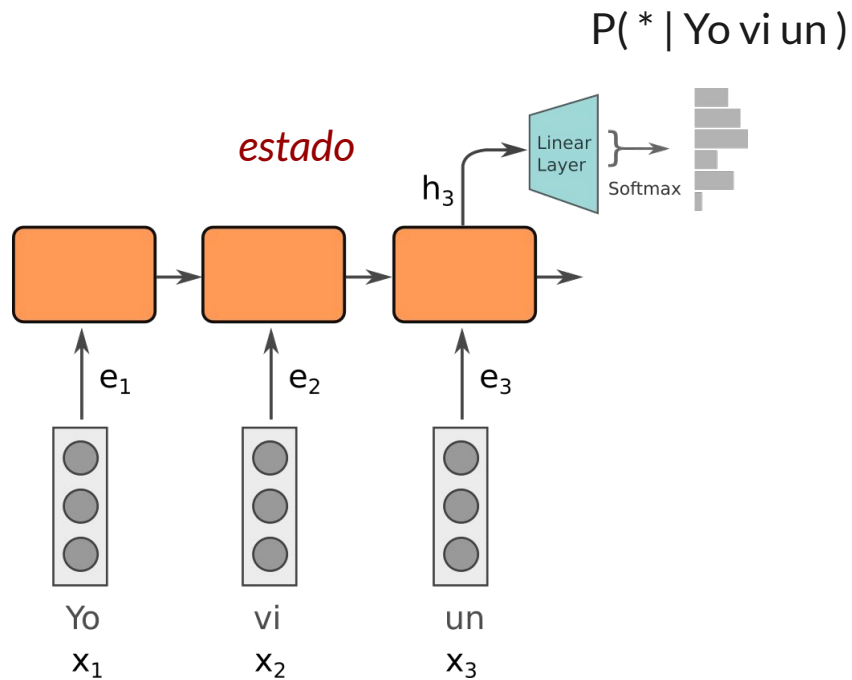
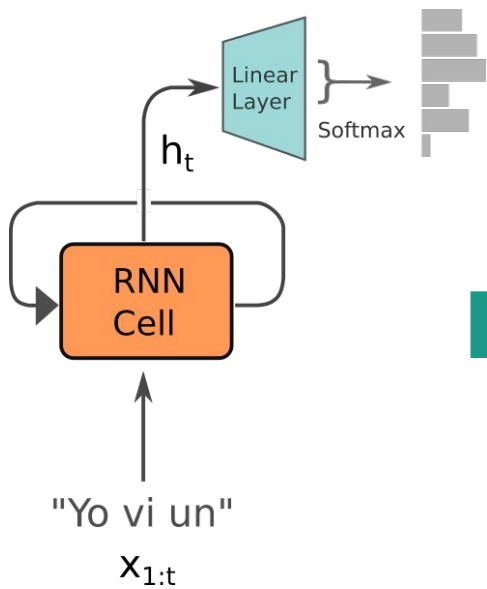


Redes Recurrentes Para Modelado de Lenguaje

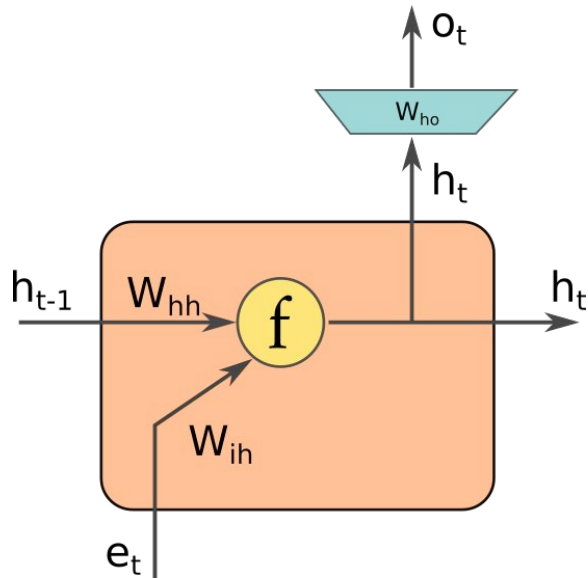
Redes Recurrentes (RNNs)



Redes Recurrentes (RNNs)



Redes Recurrentes (RNNs)



Celda Recurrente

$$h_t = f(W_{ih}e_t + W_{hh}h_{t-1} + b_h)$$

$$o_t = g(W_{ho}h_t + b_o)$$

Usualmente: $f=\tanh$, $g=\text{softmax}$

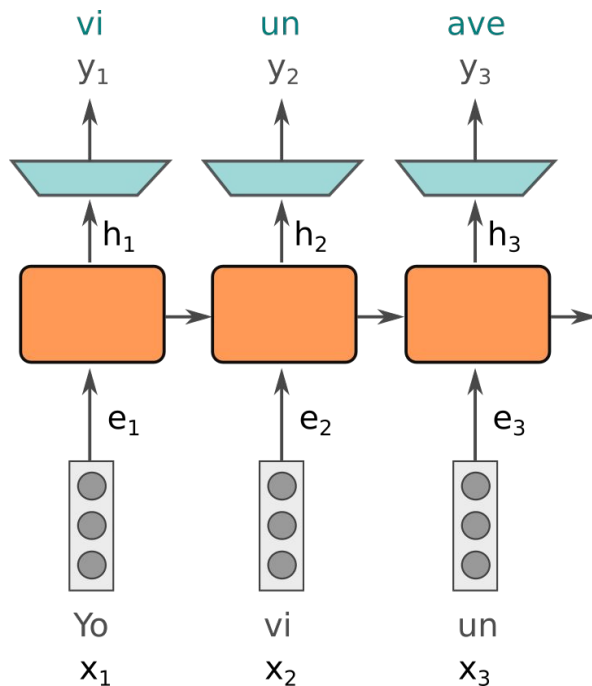
Luego, la probabilidad de token i es

$$P(y_t = i | x_{1:t}; \theta) = o_t^i$$

h_t : hidden state / estado

Otras celdas: LSTM, GRU,...

Entrenando una RNN



Loss: Cross-Entropy (CE)

En cada paso t , calcula $CE(o_t, y_t)$

Cross-Entropy entre la distribución predecida o_t y la verdadera distribución y_t (one-hot de x_{t+1})

$$J(\theta) = -\frac{1}{T} \sum_t \sum_i^{|V|} \underbrace{y_t^i}_{\text{red arrow}} \cdot \log o_t^i$$

→ y_1 : 1 para $w="vi"$
0 para el resto

$$= -\frac{1}{T} \sum_t \log o_t$$


RNN: Ventajas y Desventajas

Ventajas

- Capaz de procesar secuencias de cualquier longitud
 - Los mismos parámetros W, b son usados en cada paso
 - Tamaño del modelo no depende de la longitud del input
- No hay escarsidad de historial discreto $x_{1:t-1}$
 - Historial = h_t -> en espacio vectorial
- Aun relevantes hoy en día con el auge de State-Space-Models (ej. Mamba)

Desventajas

- ***“Vanishing Gradients”***
 - NN tiende a “olvidar” información sobre palabras lejanas a la actual t
 - LSTM, GRU son mas robustas a este fenomeno
- ***“Exploding Gradients”***
 - Las gradientes del estado h_t tienden a diverger
 - Mitigacion: gradient clipping, batch normalization



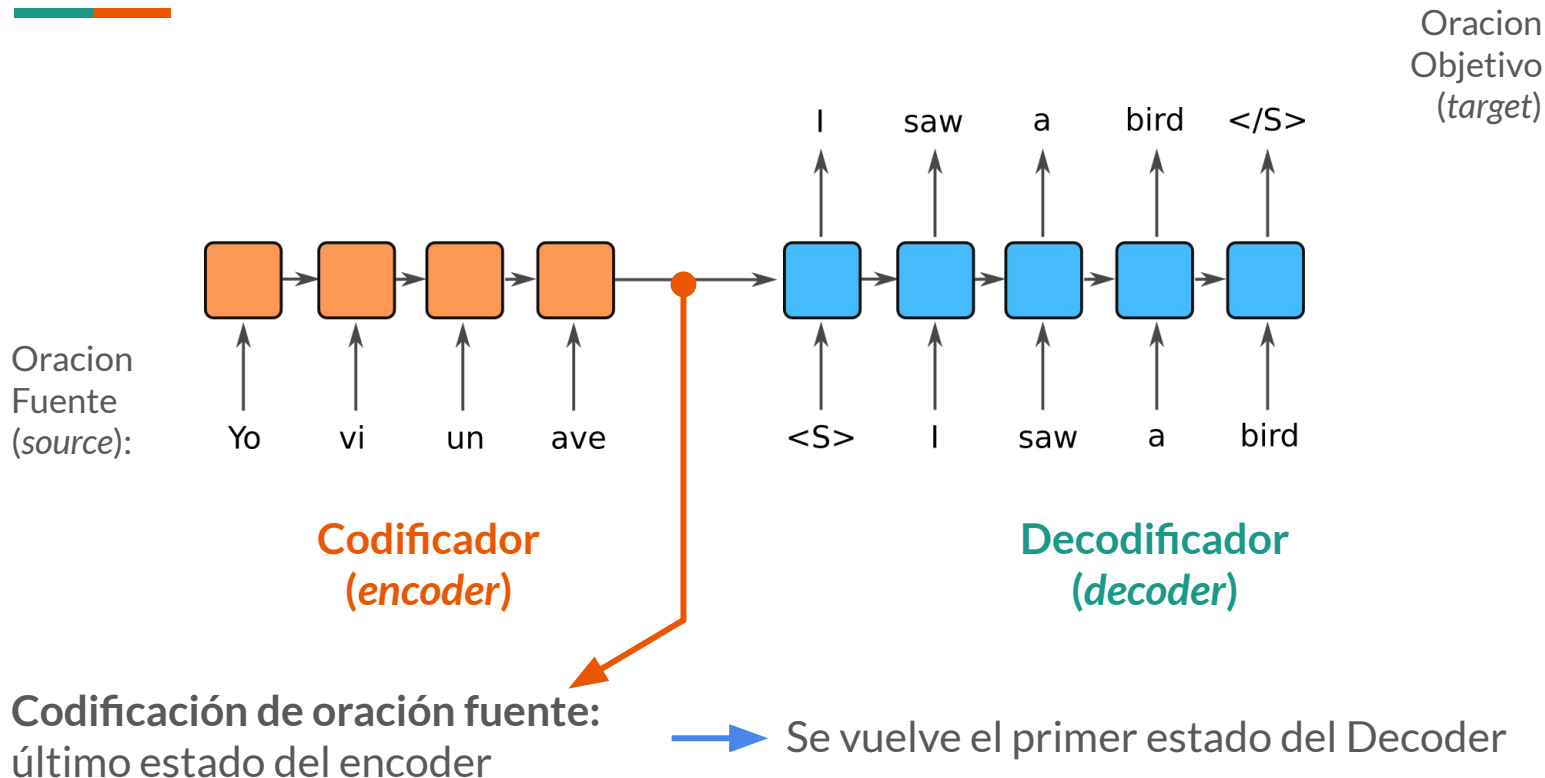
El mecanismo de Atención

Modelos Secuencia - Secuencia

El problema de “cuello de botella”

La solución: “atención”

Modelos Secuencia - Secuencia

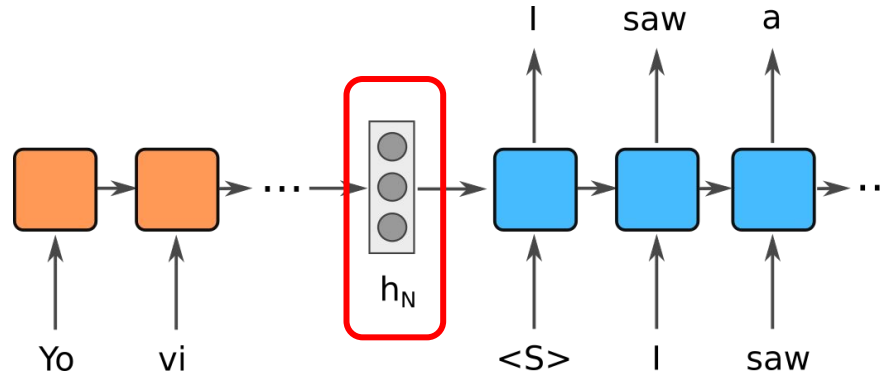


Modelos Secuencia - Secuencia



- *Sequence-to-Sequence o Seq2Seq*
- Setup efectivo para diversas tareas NLP
 - Machine Translation (traducción automática)
 - Summarization (resumen automatico)
 - Question Answering
 - ...
- LSTM llegaron a ser state-of-the-art (2014-2017)
- Arquitectura presenta una falla de diseño crucial

Problema: el “cuello de botella”



h_N : codificación de oración fuente incluir

- Debe incluir información semántica y sintáctica de N palabras
- Mientras mayor N , más información que codificar
- Dimensión de vector de estado h es fija
 - el “**cuello de botella**”
 - Inevitable : pérdida de información

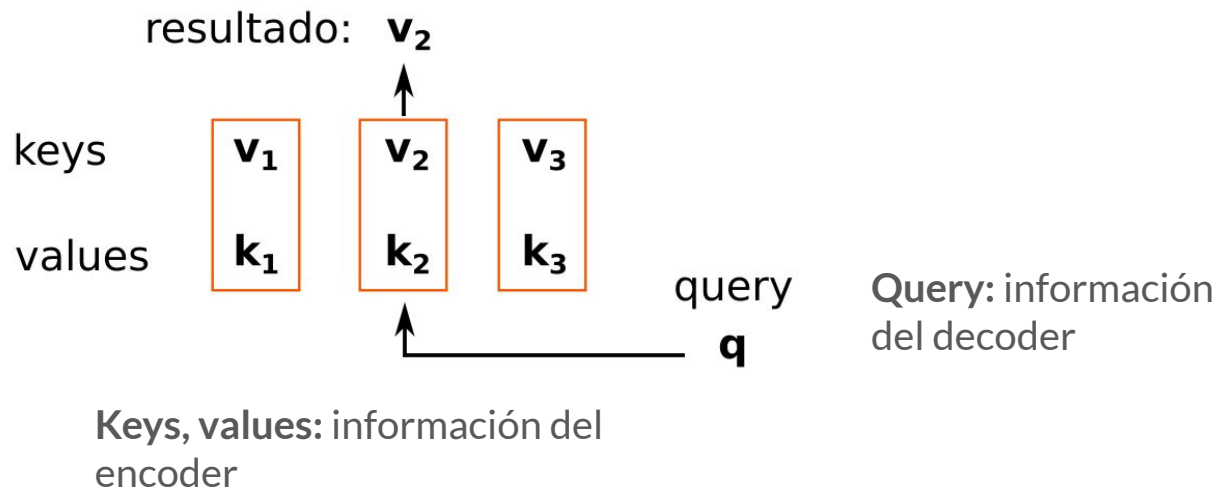
Mecanismo de atencion



- Provee una solución al problema de cuello de botella
- Idea principal:
 - En cada paso del decoder, usa información de *todos los tokens en el encoder*
 - Enfoque en una parte específica del source

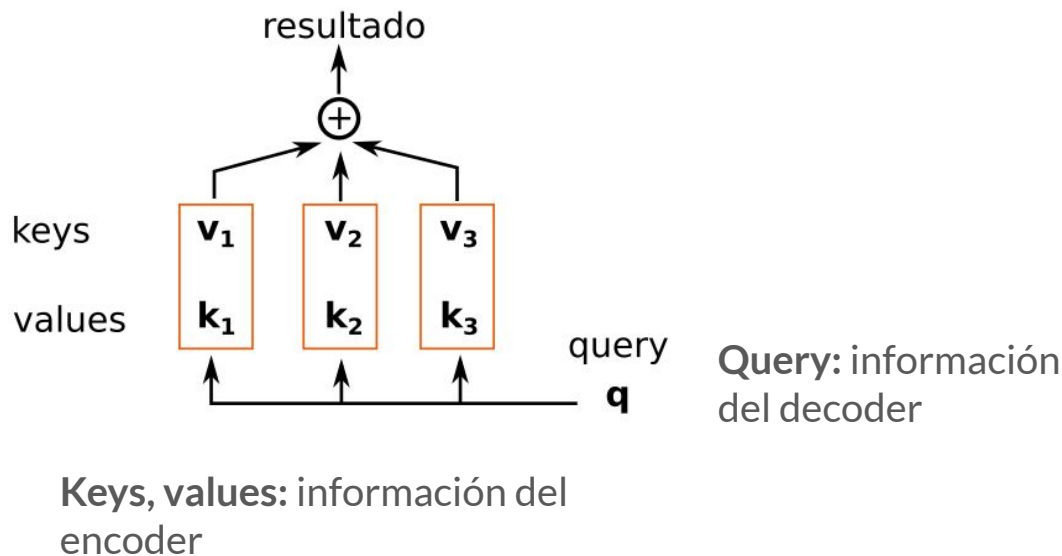
Mecanismo de atencion

- Similitud con un **Lookup Table**
 - Query hace match con **solo un solo key**



Mecanismo de atencion

- Atención = promedio ponderado de *todos los values*
 - Pesos = similitud entre **query** y cada **key**
 - **Alinea** el query con la oracion source

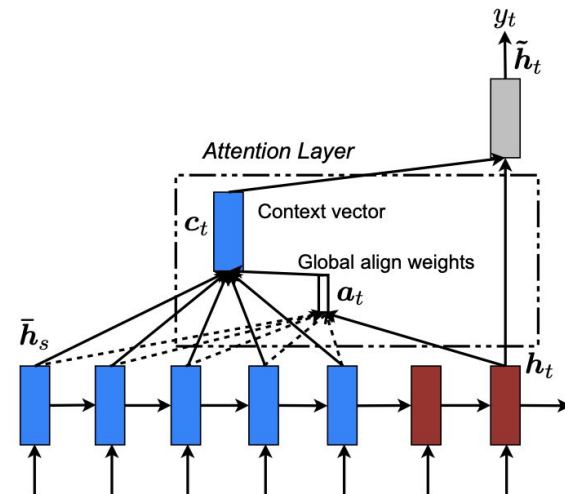


Modelo Seq2Seq con Atencion

- En paso t , tenemos el estado del decoder h_t = query
 - Calculamos la similitud con cada estado del encoder h_s = key

$$\begin{aligned} a_t(s) &= \text{align}(h_t, \bar{h}_s) \\ &= \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))} \end{aligned}$$

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \text{dot} \\ h_t^\top W_a \bar{h}_s & \text{general} \\ v_a^\top \tanh(W_a[h_t; \bar{h}_s]) & \text{concat} \end{cases}$$

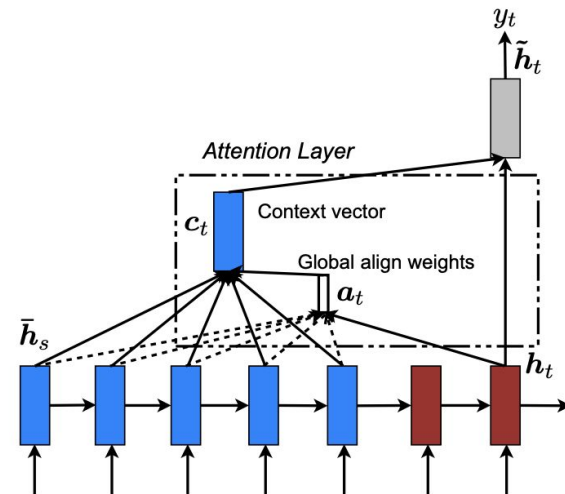


h_s : estado encoder de token s
 h_t : estado decoder de token t
 a_t : vector de alineación

Modelo Seq2Seq con Atencion

- En paso t , tenemos el estado del decoder $h_t = \text{query}$
 - Calculamos la similitud con cada estado del encoder $h_s = \text{key}$
 - Calculamos el *vector contextual* c_t , promedio ponderado de los estados encoder $h_s = \text{value}$

$$c_t = \sum_s a_t(s) \cdot \bar{h}_s$$



h_s : estado encoder de token s
 h_t : estado decoder de token t
 a_t : vector de alineación

Modelo Seq2Seq con Atencion

- En paso t , tenemos el estado del decoder $h_t = \text{query}$

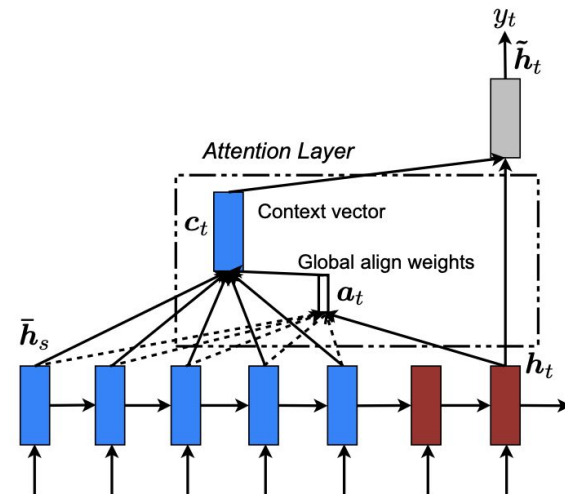
...

- Concatenamos c_t y h_t + func. de activacion

$$\tilde{h}_t = \tanh(W_c[c_t; h_t])$$

- Finalmente, la probabilidad de token y_t esta definida como

$$p(y_t|y_{<t}, x) = \text{softmax}(W_s \tilde{h}_t)$$

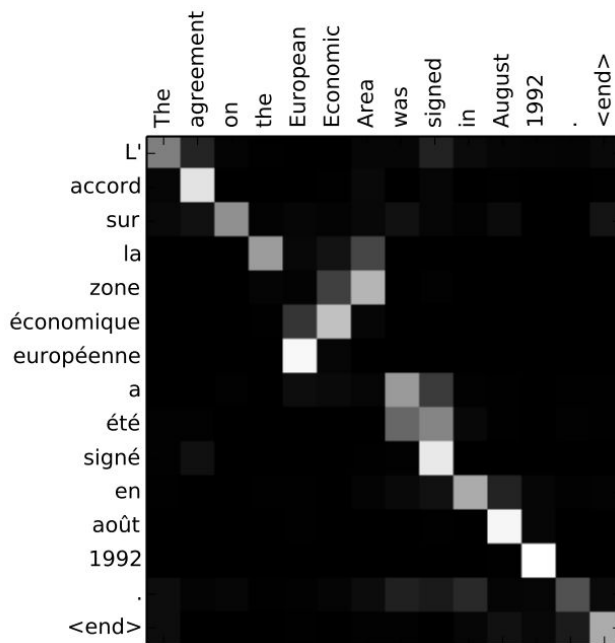


h_s : estado encoder de token s

h_t : estado decoder de token t

a_t : vector de alineación

Atencion: alineacion source-target



Atencion: ventajas y logros



- Paralelizable -> altamente optimizable con GPUs
- Crucial en Neural Machine Translation y ahora, en Transformers
- Funcionamiento más intuitivo cognitivamente, similar a como “humanos procesan texto”
- Resuelve el problem de cuello de botella
 - Estados decoder puede acceder a todos los estados encoder
- Interpretabilidad gratis (*debatible*)
 - El modelo aprende un alineamiento entre source - target
- Desventaja
 - Complejidad en tiempo: $O(N*M)$
 - N: |source|, M: |target|

Atencion: mas recursos



<https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

<https://web.stanford.edu/class/cs224n/index.html>



Transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer.



Transformers

Arquitectura paso a paso

Self-Attention

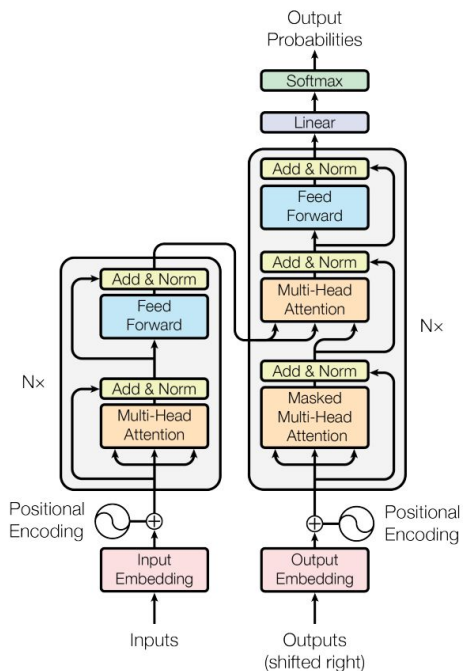
Codificando posición

Conexiones Residuales

Multi-Head Attention

Decoder

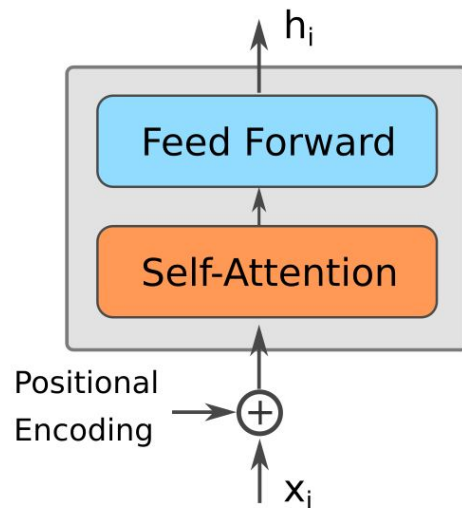
Transformers: Arquitectura



- Primera version: **encoder-decoder + atencion**
 - SOTA en Machine Translation
- Cada capa consta de varios componentes
 - Modulo de atencion
 - Normalizacion & conexiones residuales
 - Modulos lineales (FF)

Transformer Encoder: Paso a Paso

- x_i : word embedding de w_i
- h_i : estado a utilizarse en siguiente capa

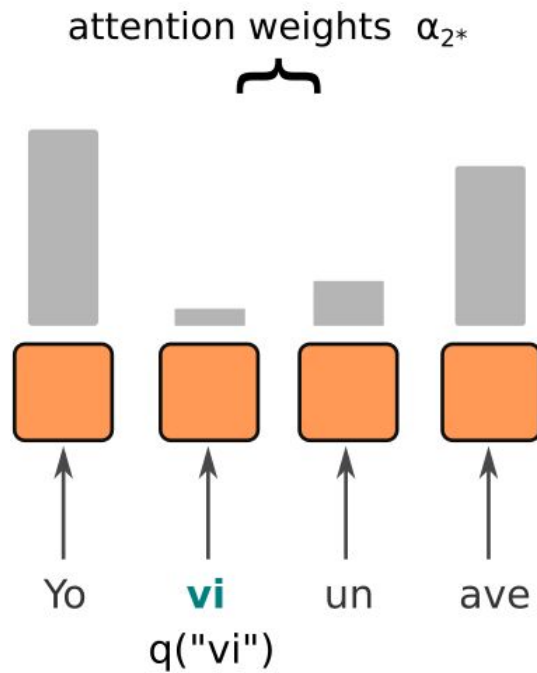


Self-Attention



- Previamente:
 - **decoder estate** -> atiende a -> **encoder states**
 - **Cross** - attention
- **RNN** codificando secuencia **x**
 - x_i depende de recurrencias, de historial $x_{<i}$
 - Recurrencias *no son paralelizables*
- Mecanismo de atencion
 - x_i toma información de todo el resto de la secuencia $x_j, j \neq i$
 - **Paralelizable!**

Self-Attention



Self-Attention

- Sea $\mathbf{x}_{1:N}$ la secuencia de word embeddings de oracion $w_{1:N}$
- Obtenemos vectores query \mathbf{q} , keys \mathbf{k} , values \mathbf{v}

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

Donde $W^* \in \mathbb{R}^{d \times d}$ son los parámetros de proyección, d =dimension

- Calculamos similitudes query-key, normalizamos con softmax

$$e_{ij} = \mathbf{q}_i^\top \mathbf{k}_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

- Calculamos la representación de w_i como el promedio ponderado

$$\mathbf{o}_i = \sum_j \alpha_{ij} \mathbf{v}_j$$

Self-Attention & codificando posición



Problema

- Self-attention no está diseñado para codificar la posición global / local de cada w_i
- o_i solo representa un “bag-of-words”

Solucion

- Agregar información posicional en los embeddings, x_i

$$\tilde{x}_i = x_i + p_i$$

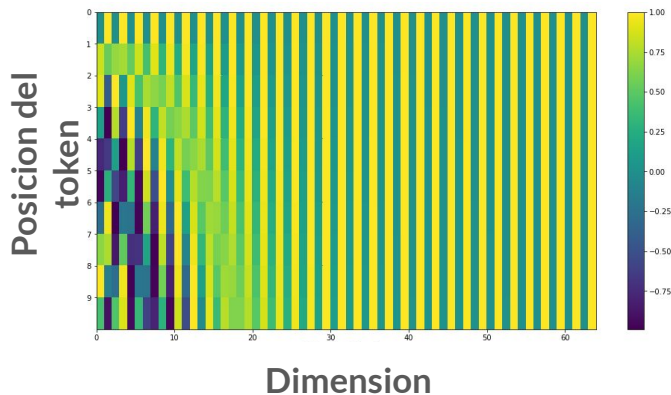
Donde $p_i \in \mathbb{R}^d$, for $i \in \{1, 2, \dots, n\}$ son vectores posicionales

Position Encodings

Representacion sinusoidal (Vaswani et al, 2017)

- Concatena funciones sinusoidales de varios periodos

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



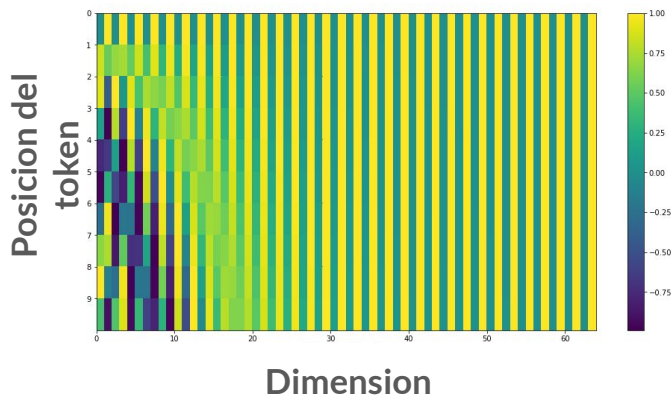
- Ventajas:
 - Periodicidad indica que la *posición absoluta* no es tan importante
 - Posible extrapolar a secuencias más largas

Position Encoding

Representacion sinusoidal (Vaswani et al., 2017)

- Concatena funciones sinusoidales de varios periodos

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Ventajas:
 - Periodicidad indica que la *posición absoluta* no es tan importante
 - Posible extrapolar a secuencias más largas

Transformer Encoder: Paso a Paso

En resumen, hasta ahora, nuestro encoder

- Agrega información posicional a x_i

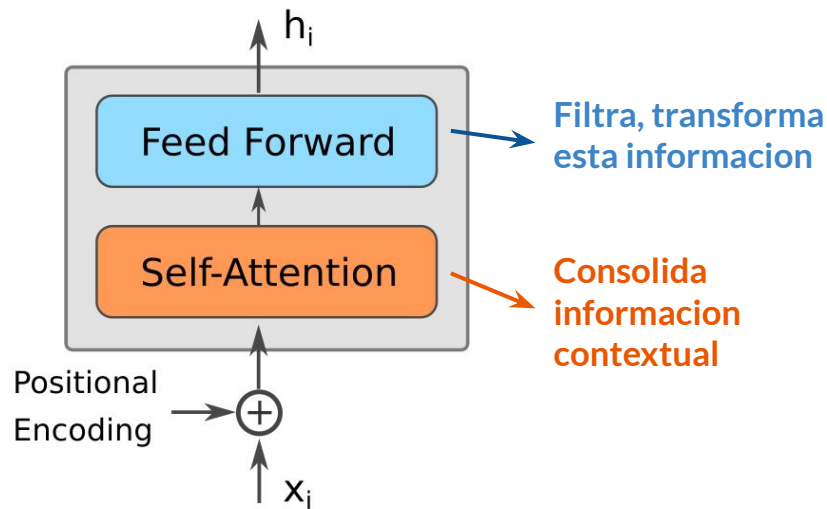
$$\tilde{x}_i = x_i + p_i$$

- Aplica self-attention y la capa lineal

$$o_i = \text{Self-Attention}(\tilde{x}_i)$$

$$h_i = g(W \cdot o_i + b)$$

$g(*)$: ReLU, GeLU,...



x_i : word embedding de w_i

h_i : estado a utilizarse en siguiente capa

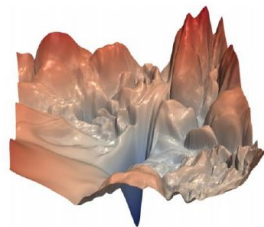
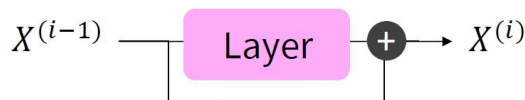
Conexiones Residuales (He et al., 2016)

Problema

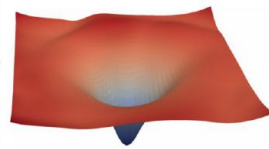
- A medida que num. de capas aumenta, h_i se encoge
- Las gradientes desvanecen (vanishing gradients de nuevo!!)

Solucion

- Hacer que NN aprenda solo la diferencia entre estados de capas, el “residual”
- Estabiliza grandemente el entrenamiento de NN profundas



[no residuals]



[residuals]

Layer Normalization (Ba et al. 2016)

Problema

- Sin control, cada capa codifica información en h_i que no es informativa para siguientes capas

Solucion

- Hacer que h_i tenga media 0 y desviación standard 1
 - Z-normalization

$$\hat{h} = \frac{h - \mu}{\sqrt{\sigma + \epsilon}} \cdot \gamma + \beta$$

μ : mean(h)

σ : std(h)

γ : ganancia (entrenable)

β : bias (entrenable)

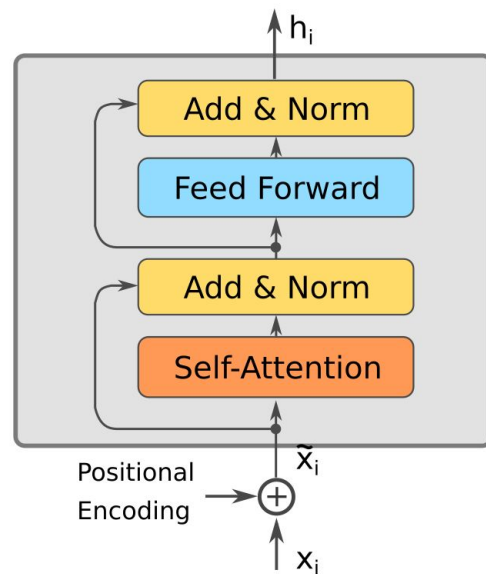
Transformer Encoder: Paso a Paso

Agregando Layer Normalization y conexiones residuales (Add & Norm), tenemos

$$o_i = \text{Self-Attention}(\tilde{x}_i)$$

$$z_i = \text{LayerNorm}(o_i) + \tilde{x}_i$$

$$h_i = \text{LayerNorm}(g(W \cdot z_i + b)) + z_i$$



x_i : word embedding de w_i

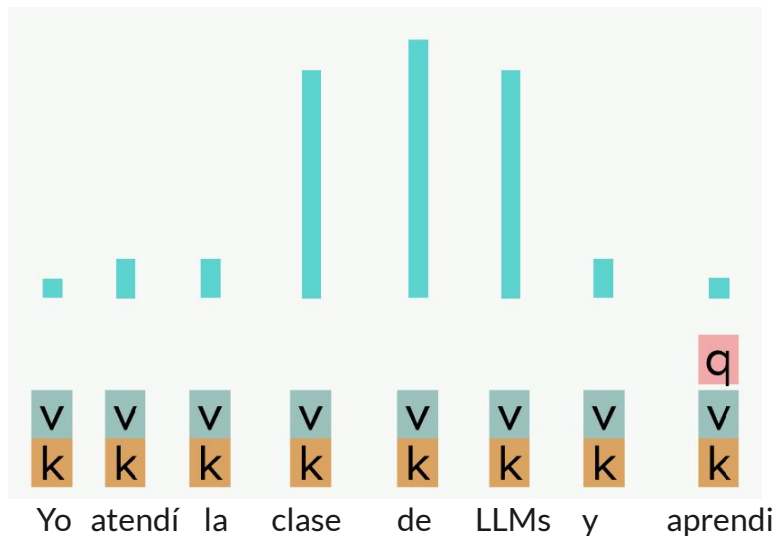
h_i : estado a utilizarse en siguiente capa

Multi-Head Attention

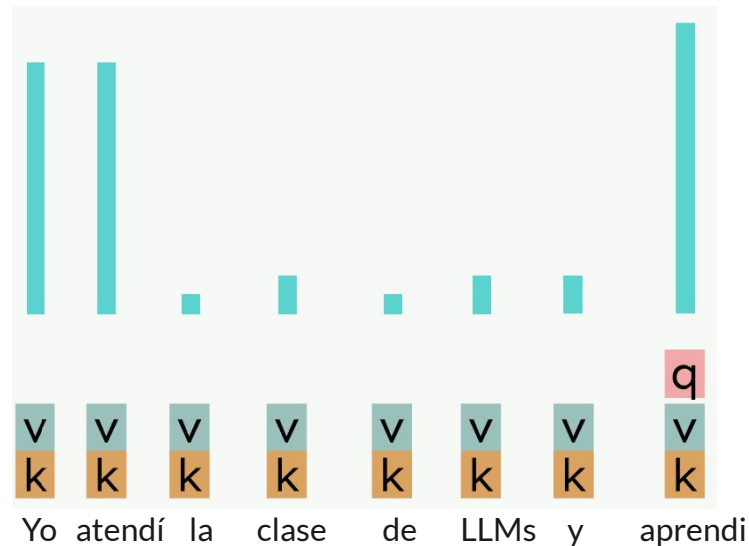


- Self-attention : un solo grupo de Q,K,V de dimensión d
 - Aprende un “único espacio de representación”, e.j.
 - Identifica sinonimos
 - Identifica entidades del mismo tipo (PER,LOC)
 - ...
- Y si queremos más granularidad?
 - Multi-head -> multiples modulos de atencion en paralelo
 - Cada módulo / head se especializa en un tipo de fenómeno

Multi-Head Attention



Head 1: especializada en frases nominales



Head 2: especializada en relaciones gramaticales (sujeto-verbo)

Multi-Head Attention

- Sea $X = [x_1, \dots, x_n]$, matrix de embeddings de todo el input, $\dim = [n \times d]$
- Para cada head m , sean q_m, k_m, v_m las matrices query, key, value

$$Q_m = QW_m^Q \quad K_m = KW_m^K \quad V_m = VW_m^V$$

donde $W_m^Q, W_m^K, W_m^V \in \mathbb{R}^{d \times d_H}$, $d_H = d/H$, $m \in [1, H]$, H =numero de heads

- Cada attention head aplica atencion independientemente

$$\text{Attention}(Q_m, K_m, V_m) = \text{softmax}\left(\frac{Q_m \cdot K_m^T}{\sqrt{d_H}}\right) V_m$$

**Scaled Dot-Product
Attention**

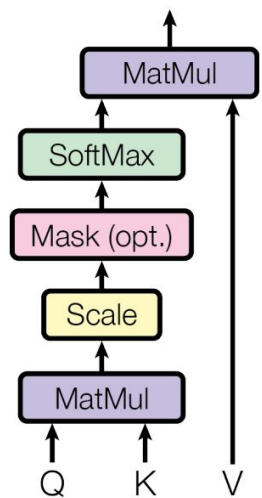
- Finalmente, Multi-head attention se define como

$$\text{Multi-Head}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_H) \cdot W^O$$

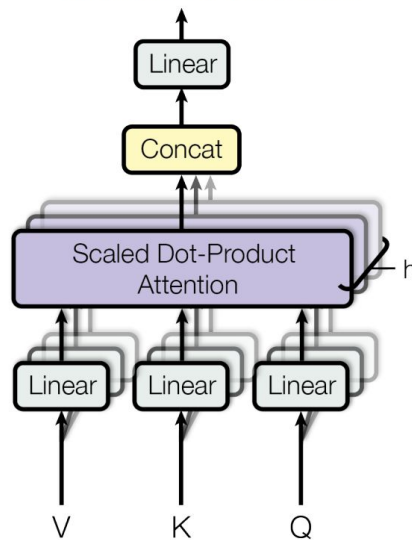
con $\text{head}_m = \text{Attention}(Q_m, K_m, V_m)$

Multi-Head Attention

Scaled Dot-Product Attention



Multi-Head Attention



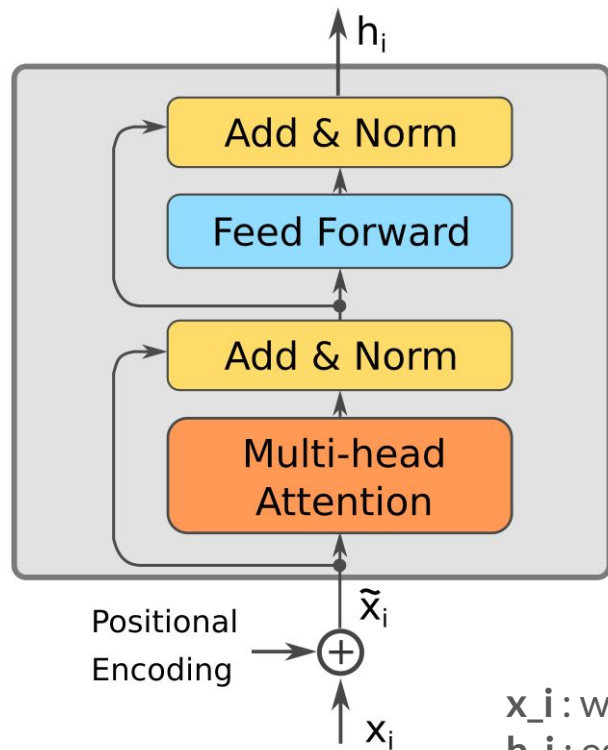
Multi-Head Attention



Ventajas

- Eficiente
 - El num. de parámetros se mantiene igual, $W^* : [d \times d] \rightarrow [d \times d/H] \times H$
 - No hay necesidad de separar W , solo reshape / transpose
- “Scaled” Dot-Product
 - El producto escalar se vuelve alto cuanto más grande d
 - Entrada del softmax se hace alto
 - Gradientes pequeñas
 - Reescalar el producto escalar estabiliza el entrenamiento

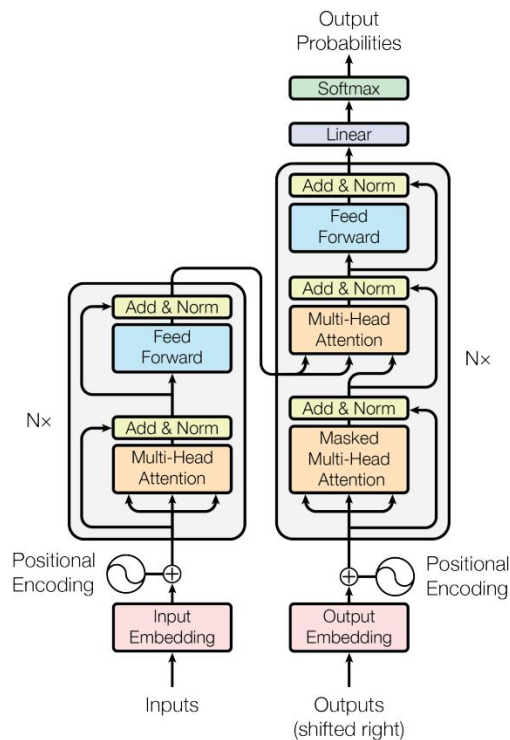
Transformer Encoder



x_i : word embedding de w_i

h_i : estado a utilizarse en siguiente capa

Transformer Decoder



Cross (Multi-head) Attention

- calculamos **Q** en base al **decoder state**
- Calculamos **K,V** en base a los **encoder states**

Masked Multi-head Attention

- En paso t , solo atendemos a $y_{1:(t-1)}$
 - K,V calculado en base a tokens previos
- Evita “ver al futuro” (y_t)

Transformer Decoder

Masked Multi-head Attention

- Sea i el decoding step (queremos generar o entrenar y_i)
- Sea j la posición a la que queremos atender

$$q_{ij} = \begin{cases} \tilde{x}_{ij} \cdot w_{ij}^Q & \text{si } j \leq i \\ -\infty & \text{si } j > i \end{cases}$$

Analogamente para k, v

- También conocido como **Causal Attention**

		j			
		<S>	Yo	vi	un
i	<S>		$-\infty$	$-\infty$	$-\infty$
	Yo			$-\infty$	$-\infty$
	vi				$-\infty$
	un				

Q,K,V

Transformer Decoder

Masked Multi-head Attention

- Sea i el decoding step (queremos generar o entrenar y_i)
- Sea j la posición a la que queremos atender

$$q_{ij} = \begin{cases} \tilde{x}_{ij} \cdot w_{ij}^Q & \text{si } j \leq i \\ -\infty & \text{si } j > i \end{cases}$$

Analogamente para k, v

- También conocido como **Causal Attention**

		j			
		<S>	Yo	vi	un
i	<S>		$-\infty$	$-\infty$	$-\infty$
	Yo			$-\infty$	$-\infty$
	vi				$-\infty$
	un				

Q,K,V

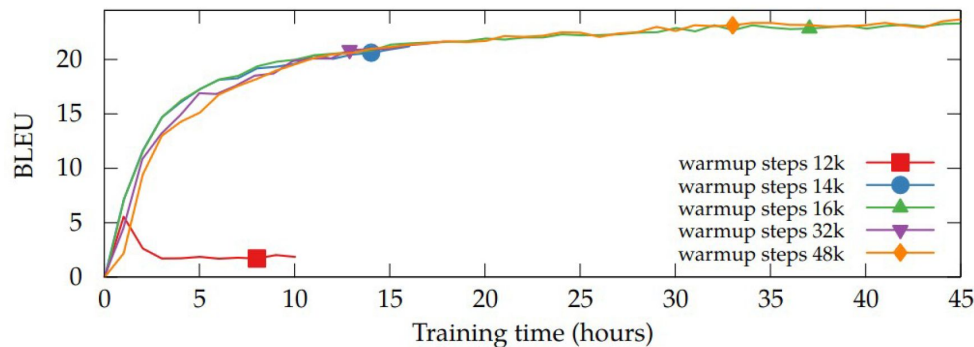
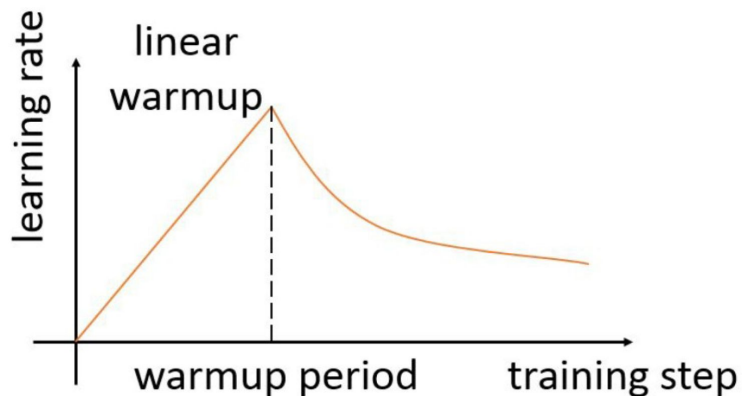
Tips: Entrenando un Transformer



- Si dispones de GPU(s), usa el batch size mas grande que te permita la memoria.
- Una vez determinado el batch size más grande posible, toca encontrar el learning rate apropiado
 - Hint: mientras más grande el modelo (en # de parametros), o mas grande el batch size, mas bajo debe ser el *learning rate*.
 - En NLP, valores entre $1e-6$ y $1e-4$ son comunes.
- Validation loss no esta correlacionado con metricas de la tarea (e.g. accuracy)
 - Quiere decir, un validation loss más bajo no necesariamente corresponde a una mejor métrica
 - En lo posible, siempre escoge el model checkpoint que muestre mejores métricas de tarea.

Tips: Entrenando un Transformer

- En vez de usar un *learning rate* (LR) constante
 - Aplica “warm-up”: aumenta linealmente el LR por N pasos



<https://arxiv.org/abs/1804.00247>

HuggingFace: demo



<https://colab.research.google.com/drive/1UOHnk-euMeIMx1KYg5nefwmPRjT4baVY?usp=sharing>

Transformer: visualizando atencion



<https://colab.research.google.com/drive/1YHhsLpFGDS7ov5td174xoJ1l86VIGd-f?usp=sharing>

<https://github.com/jessevig/bertviz>



Trabajo de Laboratorio 1

Objetivo:

- Implementar Cross-Entropy loss
- Implementar Scaled-Dot Attention

Preguntas?



References

[Tips for Presenting Your Wireframes](#)

[3 Steps to Better UI Wireframes](#)

[Wireframing for Beginners](#)