

# Thesis Title

by  
Student Name

Professor SuperProf, Advisor

A thesis submitted in partial fulfillment  
of the requirements for the  
Degree of Bachelor of Arts with Honors  
in Physics

WILLIAMS COLLEGE  
Williamstown, Massachusetts  
15 de septiembre de 2021

# Abstract

Your abstract will summarize your thesis in one or two paragraphs. This brief summary should emphasize methods and results, not introductory material.

# Executive Summary

Your executive summary will give a detailed summary of your thesis, hitting the high points and perhaps including a figure or two. This should have all of the important take-home messages; though details will of course be left for the thesis itself, here you should give enough detail for a reader to have a good idea of the content of the full document. Importantly, this summary should be able to stand alone, separate from the rest of the document, so although you will be emphasizing the key results of your work, you will probably also want to include a sentence or two of introduction and context for the work you have done.

# Acknowledgments

The acknowledgment section is optional, but most theses will include one. Feel free to thank anyone who contributed to your effort if the mood strikes you. Inside jokes and small pieces of humor are fairly common here ...

# Índice general

<b>Abstract</b>	<b>I</b>
<b>Executive Summary</b>	<b>II</b>
<b>Acknowledgments</b>	<b>III</b>
<b>1. Introduction</b>	<b>1</b>
<b>2. Transformers</b>	<b>2</b>
2.1. De RNN's a Transformers . . . . .	2
2.1.1. Redes Neuronales Recurrentes más comunes . . . . .	3
2.1.2. Compuertas LSTM y GRU . . . . .	6
2.1.3. Mecanismos de Atención . . . . .	9
2.2. El modelo Transformer . . . . .	16
2.2.1. El Codificador y Decodificador . . . . .	16
2.2.2. Multi-Head Self-Attention . . . . .	17
2.2.3. Información Posicional . . . . .	20
2.2.4. Problemas típicos en el entrenamiento de Transformers . . . . .	21
<b>A. An appendix</b>	<b>29</b>
A.1. About the bibliography . . . . .	29

# Índice de figuras

2.1. RNN - Grafo Computacional . . . . .	3
2.2. RNN - CFG . . . . .	4
2.3. RNN - Image Captioning . . . . .	5
2.4. Computo del estado oculto y salida de una Red Neuronal Recurrente. . . . .	7
2.5. Descripción. . . . .	8
2.6. Descripción. . . . .	10
2.7. Descripción. . . . .	10
2.8. Modelo seq2seq propuesto por Bahdanau, Cho, and Bengio con <i>Additive/Concat Attention</i> . . . . .	13
2.9. Modelo Transformer generalizado como modelo Secuencia a Secuencia . . . . .	16
2.10. Etapa Codificadora del Modelo Transformer. Pseudocódigo . . . . .	17
2.11. Etapa Decodificadora del Modelo Transformer. Pseudocódigo . . . . .	18
2.12. Esquema de entrenamiento e inferencia del modelo Transformer en un problema de Machine Translation. . . . .	19
2.13. 2000 Vectores de Positional Encoding con dimensiones de embedding=500. . . . .	20
2.14. Noam-Warmup con $warmup\_steps = 4000$ y $d_m = 512$ . . . . .	21
2.15. Learning rate sobre X 18000 iteraciones usando RAdam y lineal, exponencial warmup con Adam . . . . .	23

2.16. Histograma de gradientes del Algoritmo Adam con y sin etapa de <i>WarmUp</i> y usando inicialización <i>T-Fixup</i> . Imagen original de Huang, Perez, Ba, and Volkovs . . . . .	24
2.17. Visualización de 8 cabezas de atención sobre una tarea de Machine-Translation. Las matrices de atención tienden a ser ralas al tener carencia de relaciones relevantes entre diversas representaciones a distancias lejanas. . . . .	25
2.18. Transformer-XL. Para tratar con secuencias largas divide el proceso en secuencias más cortas creando estados ocultos intermedios y usandolos en el cálculo de las próximas secuencias. Figura obtenida de [11]. . . . .	26
2.19. Fast-Former. Remplaza la atención tradicional del transformer por una iterativa. En cada paso crea una consulta y clave global usando atención sobre estos mismos. Figura obtenida de [40] . . . . .	28

# Capítulo 1

## Introduction



# Capítulo 2

## Transformers

### 2.1. De RNN's a Transformers

Las **Redes Neuronales Recurrentes** o **RNN** (por sus siglas en Inglés) basadas en el trabajo de Rumelhart, Hinton, and Williams datan del año 1986. Este tipo de redes están especializadas en el procesamiento de datos que contienen información temporal, mejorando los resultados obtenidos por otros tipos de redes como *Redes FeedForward* o *Redes Convolucionales*.

La idea principal detrás de estos modelos de redes es el concepto de *Parameter Sharing*. Usando *Parameter Sharing* un modelo puede generalizar mejor cuando la información está contenida en diferentes partes de una secuencia. Así, el modelo no necesita aprender independientemente todas las reglas que forman la secuencias, sino que ahora, la salida para cada elemento perteneciente a un tiempo  $t$  está determinada por la salida del elemento anterior  $t - 1$ . Resultando en una recurrencia con las mismas reglas de actualización aplicadas a cada elemento en el tiempo. La ecuación 2.1 representa este proceso;  $h^{(t)}$  es el estado de la recurrencia definida por una función  $f$  sobre un elemento  $x^{(t)}$  de la secuencia  $X$  en el tiempo  $t$  y  $\theta$  son los parámetros compartidos.

$$h^{(t)} = f(x^{(t)}, h^{(t-1)}; \theta) \quad (2.1)$$

En una *RNN* vista como un *gráfo computacional dirigido y acíclico*, cada nodo representa un estado en la recurrencia y procesa la información de la secuencia  $X$  con los mismos parámetros  $\theta$  en cada paso, observe la figura 2.1.

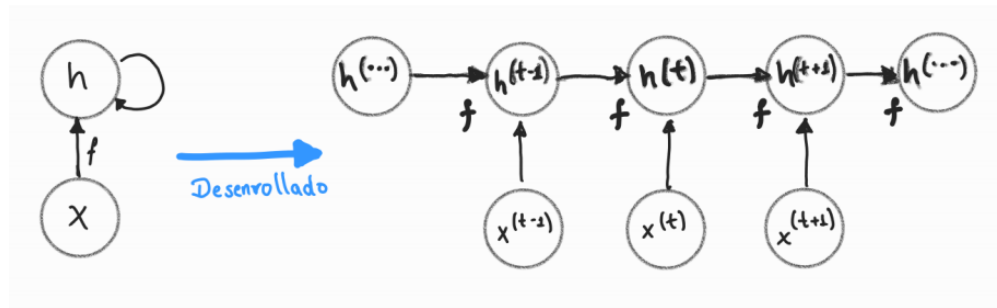


Figura 2.1: Grafo computacional generado por una *RNN* al “desenrollar” la recurrencia. Usando los parámetros compartidos en cada nodo y con cada elemento  $x^{(t)}$  de la secuencia genera un nuevo estado oculto  $h^{(t)}$  para retroalimentar nuevamente la entrada del siguiente nodo.

### 2.1.1. Redes Neuronales Recurrentes más comunes

Existen diversas formas como construir *Redes Neuronales Recurrentes*, estas pueden producir una salida en cada paso de tiempo o tener solo una al final de la recurrencia o tener conexiones entre unidades ocultas. La manera más común de implementar una *RNN* está ilustrada en la figura 2.2a. En esta figura, cada etapa de la recurrencia es retroalimentada por la activación del estado oculto previo. Así,  $h^{(t)}$  contiene información codificada de elementos previos de la secuencia que puede ser usada en el futuro para obtener una salida  $O^{(t+1)}$ . En la figura 2.2b se cambia la retroalimentación de  $h^{(t)}$  por  $o^{(t)}$ . Nótese que en este caso, la red es entrenada para obtener un valor en específico  $o^{(t)}$  lo que provocaría que gran parte de la información de los estados ocultos pasados  $h^{(t-1)}, h^{(t-2)}, \dots$  no se transmita. La diferencia entre los dos esquemas anteriores es que la red 2.2a es entrenada para decidir que información debe transmitir en el futuro a través de los estados ocultos, en cambio, en la figura 2.2b cada estado está conectado con el pasado a través de la predicción del paso anterior, perdiendo así gran parte de la información codificada en cada estado oculto  $h^{(t)}$ . Este no sería un problema si la salida  $O^{(t-1)}$  fuese lo suficientemente enriquecedora y en altas dimensiones.

Por otro lado, la *RNN* representada en la figura 2.2c tiene una sola salida al final de la recurrencia. Al contrario de las anteriores, este tipo de redes pueden ser usadas para resumir información contenida en la secuencia para finalmente predecir un único valor final. El *Análisis de Sentimiento* en textos es una tarea común que puede ser representada con este esquema de red. En la figura 2.2d vemos un modelo de *RNN* entrenado mediante el proceso de Teacher Forcing; durante el entrenamiento la red es retroalimentada con las salidas esperadas del modelo  $y^{(t)}$  en el tiempo  $t+1$ . La ventaja de esta red es que al ser eliminadas las conexiones entre estados ocultos, las funciones de pérdida basadas en comparar la predicción en el tiempo  $t$  con el valor objetivo  $y^{(t)}$  pueden ser desacopladas. Por tanto, el entrenamiento puede ser paralelizado al calcular el gradiente para cada tiempo  $t$  por separado, puesto que ya tenemos el valor ideal para esta salida.

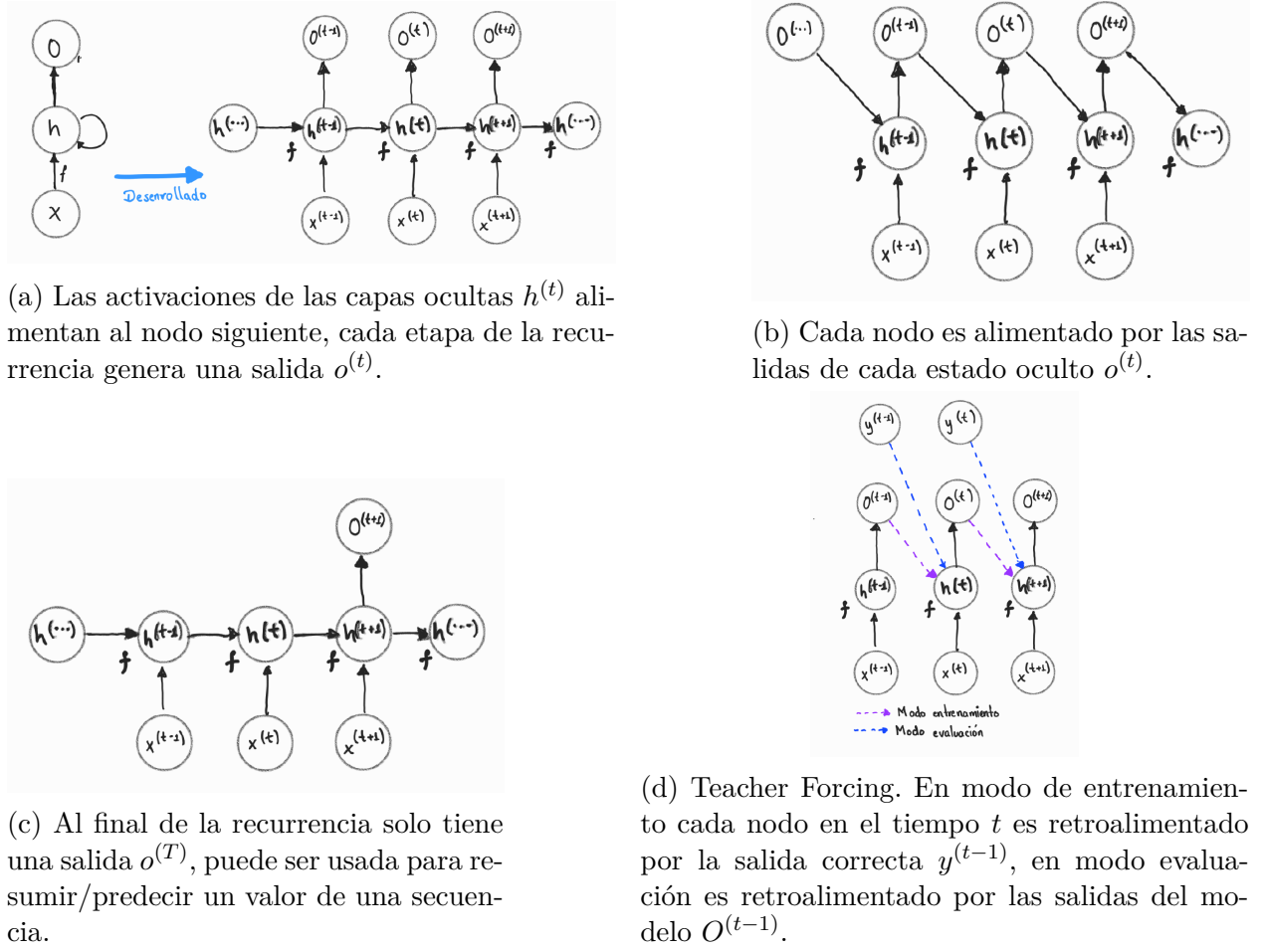


Figura 2.2: Distintos tipos de RNNs.

Finalmente, en la figura 2.3 la *Red Neuronal Recurrente* es modificada para esta vez no procesar una secuencia, sino, procesar un solo vector en cada paso. El estado oculto previo  $h^{(t-1)}$  retroalimenta al siguiente paso  $t$  así como la predicción esperada  $y^{(t)}$ , que a su vez, es usada para calcular la función de costo del paso anterior  $L^{(t-1)}$ . Esta estructura de red puede ser implementada en tareas como *Image Captioning*, en donde la entrada es una imagen y la salida una secuencia de palabras que describen esta misma.

Los modelos ejemplificados anteriormente son construidos de forma *causal*, es decir, la secuencia es procesada en un solo sentido en donde la información pasada es transmitida hacia estados futuros. Sin embargo, este flujo de información puede ser insuficiente para resolver todas las tareas. En *Modelo de Lenguaje* se aprende la estructura estadística del lenguaje con el que fue entrenado y su meta es predecir la siguiente palabra, n-grama o letra dado un contexto antes visto. En otros términos, dada una secuencia de texto de longitud  $T$   $x^{(1)}, x^{(2)}, \dots, x^{(T)}$  con  $x \in \mathcal{R}^{1 \times d}$  donde  $d$  es la dimensión de la codificación de las palabras, la

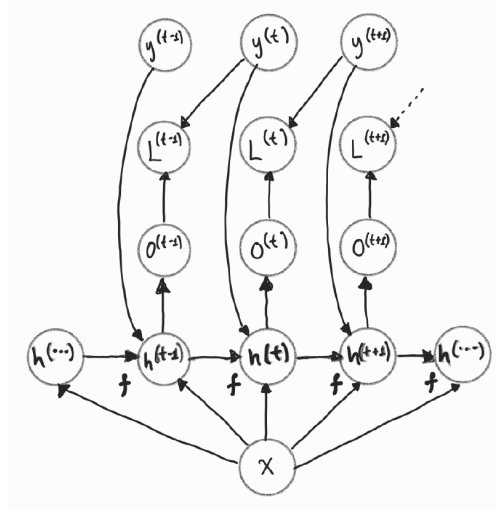


Figura 2.3: Modelo usado para tareas de *Image Captioning*, la entrada es una sola imagen y la red predice una secuencia de palabras que describen dicha imagen. La salida esperada  $y^{(t)}$  sirve como objetivo para la función de costo del paso anterior y como entrada en cada paso.

meta es predecir la probabilidad conjunta de la secuencia:

$$P(x^{(1)}, x^{(2)}, \dots, x^{(T)}) = \prod_{t=1}^T P(x^{(t)} | x^{(1)}, \dots, x^{(t-1)}) \quad (2.2)$$

Con ello, un modelo de lenguaje basado en *Redes Neuronales Recurrentes* es capaz de predecir un siguiente elemento  $\hat{x}^{(t)}$  simplemente obteniéndolo de la secuencia mediante:

$$\hat{x}^{(t)} \approx P(x^{(t)} | x^{(1)}, \dots, x^{(t-1)}) \approx P(x^{(t)} | h^{(t-1)}) \quad (2.3)$$

donde  $h^{(t-1)}$  es el estado oculto que almacena la información pasada hasta el tiempo  $t$  tal y como se definió en 2.1.

Sin embargo, la información previa de la secuencia codificada en  $h^{(t)}$  no siempre contiene los elementos necesarios para que el modelo pueda predecir correctamente el siguiente elemento, observe la siguiente oración:

« Ella estaba muy \_\_\_\_\_, después de que Alejandra vió el amanecer en la playa »

En la oración anterior, el espacio en blanco puede ser completado con algún adjetivo calificativo; *contenta*, *enojada*, *maravillada*, etc. Gracias a la información provista por la parte final de la oración, podemos deducir que de las 3 opciones la menos probable de elegir

es *enojada*. Es decir, usamos información del futuro que no pudo haber sido vista por una red (que procesa la información en forma causal) para tomar la mejor elección. Una ligera modificación fácilmente aplicable a estos modelos es que las secuencias sean procesadas en ambas direcciones, las **Redes Neuronales Recurrentes Bidireccionales** [32].

Una *RNN Bidireccional* procesa la secuencia en ambos sentidos (una *RNN* en un sentido y otra en el otro), capturando información del pasado en el estado oculto  $\vec{h}^{(t)}$  cuando la recurrencia es del inicio al final de la secuencia e información del futuro en  $\overleftarrow{h}^{(t)}$  cuando la recurrencia es del final al inicio de la secuencia. Finalmente, el estado oculto  $h^{(t)}$  es una concatenación de ambos estados  $\vec{h}^{(t)}$  y  $\overleftarrow{h}^{(t)}$ , vea la ecuación 2.4. Por lo cual, la salida  $o^{(t)}$  ahora puede ser calculada con información tanto del futuro como del pasado 2.5.

$$\begin{aligned}\vec{h}^{(t)} &= f(x^{(t)}, \vec{h}^{(t-1)}; \theta_f) \\ \overleftarrow{h}^{(t)} &= f(x^{(t)}, \overleftarrow{h}^{(t+1)}; \theta_b) \\ h^{(t)} &= \text{Concat}(\vec{h}^{(t)}, \overleftarrow{h}^{(t)})\end{aligned}\tag{2.4}$$

$$o^{(t)} = g(h^{(t)}; \theta_{out})\tag{2.5}$$

### 2.1.2. Compuertas LSTM y GRU

Hasta el momento, se ha hecho mención de las salidas  $o^{(t)}$  y estados ocultos  $h^{(t)}$  solo como el resultado de operaciones aplicadas por dos funciones;  $g$  y  $f$  respectivamente. Existen varias alternativas de construir una *RNN*, una de las maneras más comunes es usando 2.6 y 2.7:

$$h^{(t)} = \phi(x^{(t)}W_x + h^{(t-1)}W_h + b)\tag{2.6}$$

$$o^{(t)} = x^{(t)}W_{out} + b\tag{2.7}$$

Donde los parámetros compartidos de la red ahora son descritos por las matrices  $W_x \in \mathbb{R}^{d \times k}$ ,  $W_h \in \mathbb{R}^{k \times k}$  y  $W_{out} \in \mathbb{R}^{k \times q}$  con  $k$  como la dimensión del estado oculto,  $q$  la dimensión de las salidas  $o^{(t)}$ ,  $b \in \mathbb{R}^{1 \times q}$  el parámetro de sesgo y  $\phi$  es la función de activación. De esta manera, los pesos de los parámetros aprendidos en la matriz  $W_h$  determinan cómo será usada la información del pasado, codificada en  $h^{(t-1)}$ . Posteriormente, es incluida a la codificación de la información del tiempo actual  $t$  calculada por  $W_x$ . La figura 2.4 representa gráficamente la lógica usada para calcular los estados ocultos y las salidas de la red.

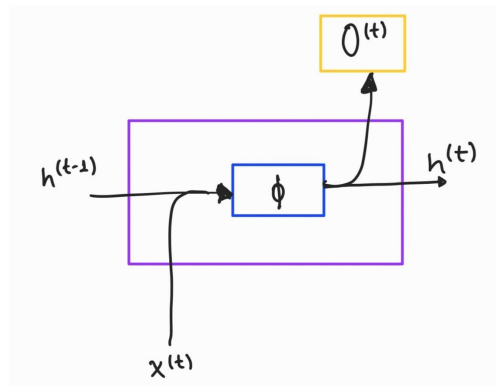


Figura 2.4: Computo del estado oculto y salida de una Red Neuronal Recurrente.

Sin embargo el cálculo de los estados ocultos mediante 2.6 presenta algunos problemas. La interacción entre la información del pasado y la actual siempre es "plana", es decir, la información fluye a través del tiempo de la misma manera sin forma de dar prioridad o ignorar parte de esta. Por lo que resulta una tarea un poco más complicada preservar información relevante a en cada paso o desechar información que ya no es útil para la red. También, causado por este mismo flujo de los datos, la información del pasado poco a poco es opacada por nueva información, impidiendo que se puedan encontrar dependencias de información en secuencias largas en tiempos distantes; comúnmente se hace referencia a este problema como *The Short-term Memory Problem* en inglés [3]. Aunado a problemas como el *Desvanecimiento o Explosión del Gradiente* [18] [28], y acentuándose aun más debido a las matrices de pesos compartidos en la recurrencia. Dichas multiplicaciones en la recurrencia tienen similitud al método de potencia, en donde cualquier componente en la matriz inicial que no esté alineada con el vector propio asociado al mayor valor propio son eventualmente descartados [13, pp. 390-392]), por ende, los resultados de este producto tendrán a ser cercanos a cero (desvanecerse) o explotar dependiendo de la magnitud de la matriz de pesos.

Una manera de solventar los problemas anteriores son las **Redes Neuronales con Compuertas**, creadas con la idea de crear conexiones a través del tiempo de tal manera de tener gradientes que no se desvanezcan o exploten, convirtiéndose además en un mecanismo para olvidar información pasada y decidiendo automáticamente cuándo y cuánto de la información debe prevalecer.

## LSTM

**Long Short-Term Memory**, **LSTM** por sus siglas en inglés, fue propuesta en 1997 por Hochreiter and Schmidhuber, como un método de preservar dependencias de información relevante distantes a corto plazo. Las *LSTM* introducen un nuevo componente la *Celda de Memoria* cuya función es guardar información a través del tiempo y es controlada por

distintas compuertas. Las compuertas aprenden a distinguir que información es relevante y cual no. Hay 3 de ellas, la *Compuerta de Entrada*, la *Compuerta de Olvido* y la *Compuerta de Salida*. La *Compuerta de Entrada*  $I^{(t)}$  (véase la figura 2.5) determina cuanta información actual debe ser contemplada a través de la *Memoria Candidata*  $\tilde{C}^{(t)}$  para actualizar la *Celda de Memoria*  $C^{(t)}$ . La *Compuerta de Olvido* indica qué información del pasado debe ser desechada de la *Celda de Memoria*  $C^{(t-1)}$  y la *Compuerta de Salida* ayuda a determinar el nuevo estado  $h^{(t)}$  via la *Celda de Memoria* actual  $C^{(t)}$ .

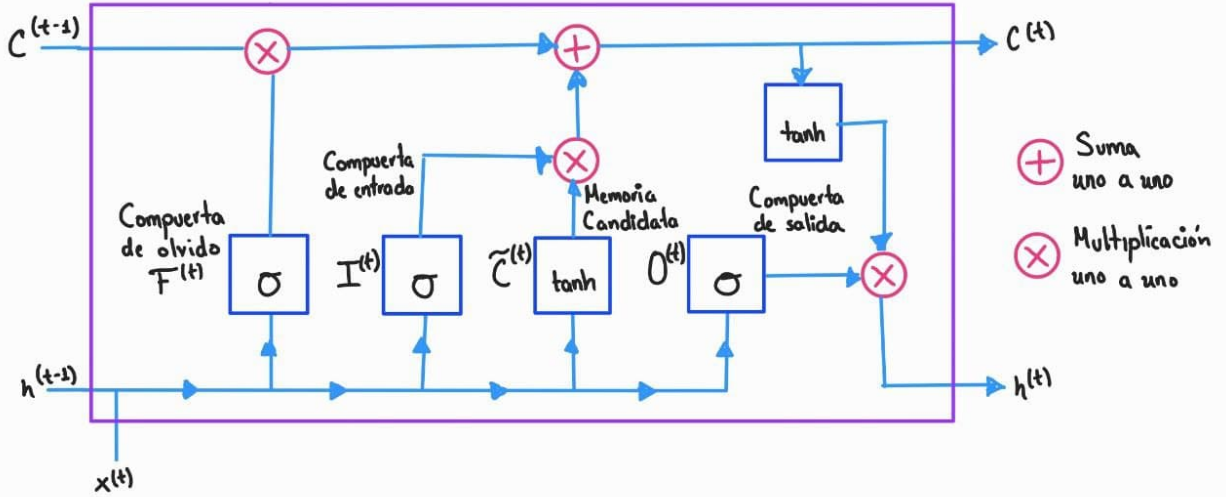


Figura 2.5: Descripción.

Las ecuaciones 2.8 rigen el comportamiento de Compuertas de Entrada, Salida y Olvido.

$$\begin{aligned} I^{(t)} &= \sigma(x^{(t)}W_{xi} + h^{(t-1)}W_{hi} + b_i) \\ F^{(t)} &= \sigma(x^{(t)}W_{xf} + h^{(t-1)}W_{hf} + b_f) \\ O^{(t)} &= \sigma(x^{(t)}W_{xo} + h^{(t-1)}W_{ho} + b_o) \end{aligned} \quad (2.8)$$

Donde  $W_{xi}, W_{xf}, W_{xo} \in \mathbb{R}^{d \times k}$ ,  $W_{hi}, W_{hf}, W_{ho} \in \mathbb{R}^{k \times k}$  y  $b_i, b_f, b_o \in \mathbb{R}^{1 \times k}$

La Memoria Candidata y la Celda de memoria son actualizadas mediante:

$$\begin{aligned} \tilde{C}^{(t)} &= \tanh(x^{(t)}W_{xi} + h^{(t)}W_{hc} + b_c) \\ C^{(t)} &= F^{(t)} \odot C^{(t-1)} + I^{(t)} \odot \tilde{C}^{(t)} \end{aligned} \quad (2.9)$$

Donde  $W_{xi} \in \mathbb{R}^{d \times k}$ ,  $W_{hc} \in \mathbb{R}^{k \times k}$  y  $b_c \in \mathbb{R}^{1 \times k}$

Y finalmente el estado oculto  $h^{(t)}$  esta dado por:

$$h^{(t)} = O^{(t)} \odot \tanh(C^{(t)}) \quad (2.10)$$

$\sigma$  y  $\odot$  denotan la función de activación sigmoide y la multiplicación uno a uno respectivamente.

## GRU

**Gated Recurrent Units** o **GRU** por sus siglas en inglés, fueron propuestas en 2014 [7] como una alternativa computacionalmente más rápida y con similar rendimiento que las *LSTM* [10]. A diferencia de anteriores, las *GRU* prescinden de la *Celda de Memoria* y utilizan un par de compuertas (la *Compuerta de Actualización* y la de *Olvido*) para decidir que información aún es necesaria que esté codificada dentro del estado oculto, véase la ecuación 2.11. La *Compuerta de Olvido* permite decidir que del pasado aún debe ser transmitido a futuros estados o de otro modo ser desechada. La *Compuerta de Actualización* indica que información nueva es relevante y necesita ser incorporada al no estar codificada dentro del estado oculto, véase la ecuación 2.12.

$$\begin{aligned} R^{(t)} &= \sigma(x^{(t)}W_{xR} + h^{(t-1)}W_{hR} + b_R) \\ Z^{(t)} &= \sigma(x^{(t)}W_{xZ} + h^{(t-1)}W_{hZ} + b_Z) \end{aligned} \quad (2.11)$$

$$\begin{aligned} \tilde{h}^{(t)} &= \tanh(x^{(t)}W_{xh} + (R^{(t)} \odot h^{(t-1)})W_{hh} + b_h) \\ h^{(t)} &= Z^{(t)} \odot h^{(t-1)} + (1 - Z^{(t)}) \odot \tilde{h}^{(t-1)} \end{aligned} \quad (2.12)$$

### 2.1.3. Mecanismos de Atención

Una de las arquitecturas comunes vistas previamente es la mostrada en la figura 2.2c cuya información procesada es resumida en una sola salida. Este tipo de red es usada como parte de las soluciones en tareas de reconocimiento de voz (*Speech Recognition*), traducción de lenguaje (*Machine Translation*) o asistencia en respuestas automáticas (*Question Answering*), entre otros, típicamente bajo modelos Secuencia a Secuencia (*Sequence to Sequence, Seq2Seq*) [8]. Los modelos *Seq2Seq* están formados por dos redes neuronales como la mostrada en 2.7. La primera se comporta como un *codificador* al resumir la entrada y producir un vector



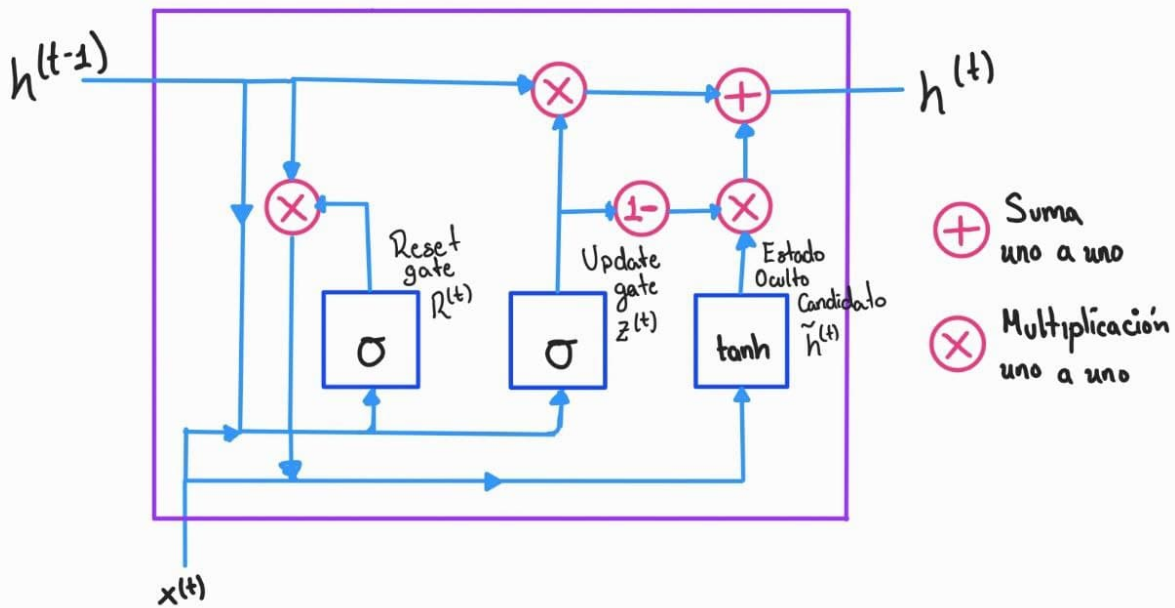


Figura 2.6: Descripción.

de salida de tamaño fijo llamado *vector de contexto*. La segunda red se comporta como un *decodificador*, este es inicializado y condicionado con el *vector de contexto* para obtener una transformación de la entrada no necesariamente del mismo tamaño de secuencia, debido a que en tareas como traducir una oración de un lenguaje a otro donde la traducción no siempre contiene las misma cantidad de palabras usadas en el idioma original.

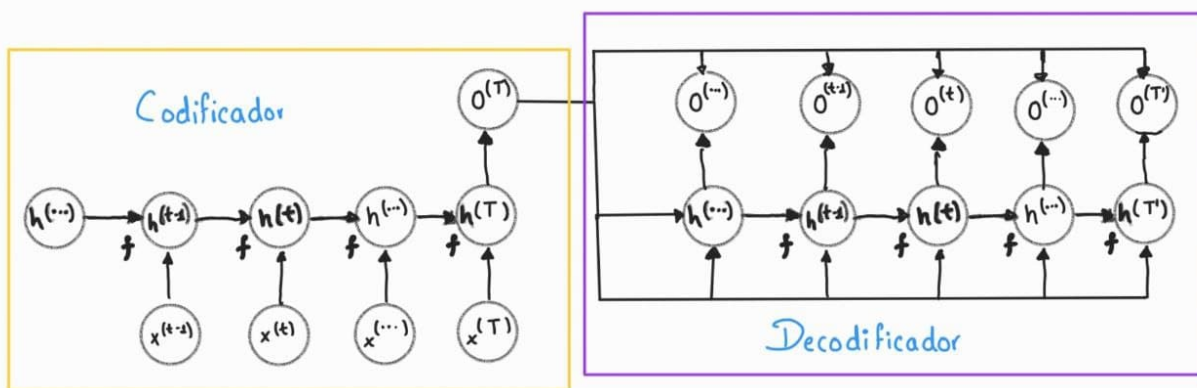


Figura 2.7: Descripción.

Por ejemplo, en tareas de *Machine Translation* el *codificador* está formado por una *RNN* Bidireccional que lee y procesa un conjunto de vectores  $X = (x^{(1)}, x^{(2)}, \dots, x^{(T_x)})$  para obtener un vector de contexto  $C$ . La forma más común es como en 2.13:

$$\begin{aligned} h^{(t)} &= f_{bi}(x^{(t)}, h^{(t-1)}; \theta_f, \theta_b) \\ C &= q(h^{(1)}, h^{(2)}, \dots, h^{(T)}) \end{aligned} \quad (2.13)$$

Recordemos que  $h^{(t)}$  es el estado oculto generado por la concatenación de los dos estados ocultos generados por la *RNN Bidireccional*,  $f_{bi}$  y  $q$  son funciones no lineales, ya sea, una *LSTM* para  $f_{bi}$  y  $q(h^{(1)}, h^{(2)}, \dots, h^{(T)}) = h^{(T)}$ , equivalente a tomar solo el ultimo estado oculto como vector de contexto  $C$ . El *decodificador* es entrenado para predecir la siguiente palabra  $y^{(t')}$  dado el vector de contexto  $C$  y todas las palabras previas predichas. En otras palabras, el decodificador define la probabilidad conjunta modelada por una *RNN*:

$$p(Y) = \prod_{t=1}^{T_y} p(y^{(t)} | \{y^{(1)}, \dots, y^{(t-1)}\}, C) \quad (2.14)$$

$$p(y^{(t)} | \{y^{(1)}, \dots, y^{(t-1)}\}, C) = g(y^{(t-1)}, s^{(t)}, C; \theta_g) \quad (2.15)$$

donde  $g$  es una función no lineal que emite la probabilidad de  $y^{(t)}$  y  $s^{(t)}$  es el estado oculto del *decodificador* 2.16.

$$s^{(t)} = f(s^{(t)}, y^{(t-1)}, C; \theta_s) \quad (2.16)$$

Sin embargo, cuando las secuencias son bastante largas el *vector de contexto* emitido por el *codificador* no es lo suficientemente grande como para resumir correctamente la secuencia y por tanto, la información inicial de la entrada es olvidada, teniendo escasa presencia en estados ocultos más lejanos. En 2015 Bahdanau, Cho, and Bengio observaron estos efectos y propusieron una forma de minimizarlos, los **Mecanismos de Atención**.

La función principal de los **Mecanismos de Atención** es permitir que el *decodificador* pueda acceder al historial completo de los estados ocultos del *codificador*, así, ahora podrá contar con un mecanismo para selectivamente centrarse en las distintas partes de la secuencia que tienen mayor influencia sobre una la salida esperada a cierto tiempo.

Por tanto, las palabras predichas no son calculadas por un único *vector de contexto* generado por el *codificador*, sino que para cada objetivo  $y^{(t)}$  se calcula un nuevo *vector de contexto*  $c^{(t)}$ :

$$p(y^{(t)} | \{y^{(1)}, \dots, y^{(t-1)}\}, c^{(t)}) = g(y^{(t-1)}, s^{(t)}, c^{(t)}; \theta_g) \quad (2.17)$$

$$s^{(t)} = f(s^{(t)}, y^{(t-1)}, c^{(t)}; \theta_s) \quad (2.18)$$

Dado que cada estado oculto  $h^{(t)}$  contiene mucho mejor la información que se encuentran alrededor del  $t$ -ésimo término, se puede generar cada vector de contexto como una suma pesada de sobre los estados ocultos del *codificador*. Estos pesos nos ayudan a determinar que tan importante es la información codificada por cada estado oculto y al momento de obtener la salida del  $t$ -ésimo valor “prestar atención” a aquellos que son más relevantes para esta predicción:

$$c^{(t)} = \sum_{i=1}^{T_x} \alpha_{t,i} h^{(i)} \quad (2.19)$$

aquí cada peso  $\alpha_{t,i}$  indica que tan bien se “alinean” los términos  $y^{(t)}$  y  $x^{(i)}$ , y son calculados por una *función de alineamiento* que denota que tan importante es el estado oculto del *codificador*  $h^{(t)}$  para el estado oculto del decodificador  $s^{(i)}$ .

$$\alpha_{t,i} = \text{align}(y^{(t)}, x^{(i)}) = \frac{\exp(\text{score}(s^{(t-1)}, h^{(i)}))}{\sum_{k=1}^{T_x} \exp(\text{score}(s^{(t-1)}, h^{(k)}))} \quad (2.20)$$

*Bahdanau* propone aprender esta alineación usando una *Red feed-forward* con una sola capa oculta y la función  $\tanh$  como activación:

$$\text{score}(s^{(t)}, h^{(i)}) = v_a^\top \tanh(W_a[s^{(t)}; h^{(i)}]) \quad (2.21)$$

con  $v_a$  y  $W_a$  como matrices de pesos a aprender durante el entrenamiento,  $[s^{(t)}; h^{(i)}]$  representa una concatenación de los estados ocultos del *codificador* y decodificador. En la figura 2.8 podemos ver gráficamente el modelo usado por *Bahdanau*.

Los modelos de atención pueden ser vistos de manera más general como un mapeo de una secuencia de llaves  $k$  hacia una distribución de atención  $\alpha$  de acuerdo a una consulta  $q$  aplicándose a un conjunto de valores  $V$  para selectivamente propagar la información contenida en  $V$ . Si bien, los términos de consulta, llaves y valores (query, keys, values) son en ámbitos de los *Sistemas de Recuperación de Información* su relación en términos de la atención aplicada por Bahdanau es muy similar; las llaves son los estados ocultos del *codificador* y la consulta es el estado oculto del decodificador en cuestión, en este caso el mapeo de entre llaves y valores es la misma:

$$A(q, K, V) = \sum_i p(a(K - i, q)) * v_i \quad (2.22)$$

En la ecuación 2.22,  $p$  es una función de distribución que mapea los puntajes de la función de alineación  $a$  a pesos de atención. Comúnmente se usan las funciones *softmax* o *logistic*

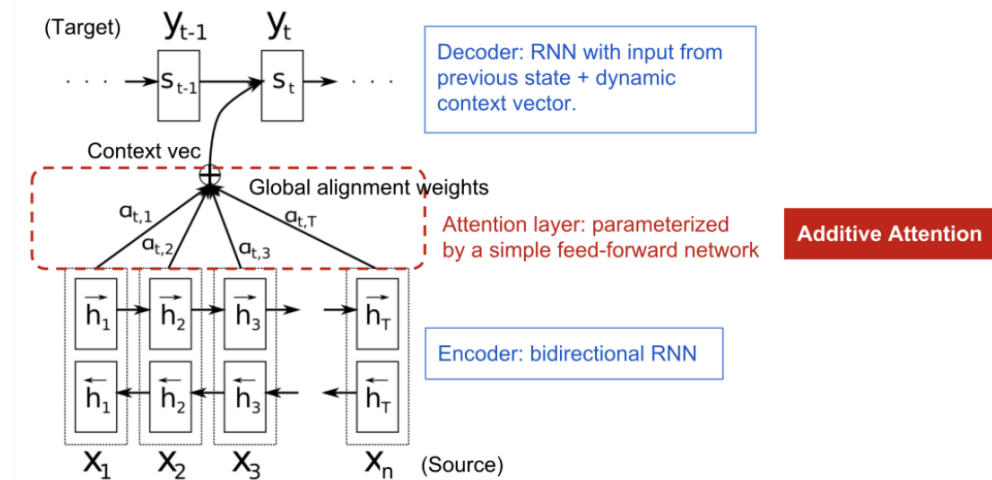


Figura 2.8: Modelo seq2seq propuesto por Bahdanau, Cho, and Bengio con *Additive/Concat Attention*

*sigmoid* puesto que nos aseguran que los pesos de atención producidos estarán dentro del rango  $[0, 1]$  y la suma de ellos es igual a 1, por lo que los pesos pueden ser interpretados como una probabilidad que indica que tan relevante es cierto elemento. Algunas variaciones en donde se consideran solo los términos relevantes como *sparsemax* Martins and Astudillo o *sparse entmax* Martins, Treviso, Farinhas, Niculae, Figueiredo, and Aguiar permiten trabajar y enfocarse en solo algunas relaciones de alineamiento. Tay, Luu, Zhang, Wang, and Hui proponen una función de distribución de pesos  $M = \tanh(E) \odot \text{sigmoid}(N)$  con  $E$  como una matrix en donde cada entrada representa la similitud entre estados ocultos y  $N$  una medida negativa (disimilitud), por lo que podemos usar  $\text{sigmoid}(N)$  como información para “de-atender” los alineamientos de  $E$ .

Las funciones de alineamiento se encargan de comparar y extraer la relación entre las representaciones de las llaves (keys) y consultas (queries), por ejemplo usando el producto punto y el coseno como función de similitud. Bahdanau calcula esta relación a través de una red neuronal 2.20, lo que evita asumir que ambas representaciones están en el mismo espacio, como lo hace las funciones de alineación como el producto punto o la similitud coseno. La tabla 2.1 muestra una recopilación de funciones de alineamiento.

De acuerdo a como es aplicado los distintos tipos de atención Chaudhari, Polatkan, Ramanath, and Mithal los dividen en 4 grandes grupos; por número de secuencias, por nivel de abstracción, por número de posiciones y por número de representaciones. Estos grupos no son mutuamente excluyentes por tanto una aplicación de atención puede pertenecer a más de una.

En la categoría *por número de secuencias* se identifican 3 tipos, el primero de ellos, **Distintivos** (*Distinctive*) es cuando la clave (key) y valor (value) pertenecen a distintas

Nombre	Función de Alineación	Cita
Similarity / Content-Base	$a(k_i, q) = \text{sim}(k_i, q)$	Graves, Wayne, and Danihelka
Dot Product <sup>1</sup>	$a(k_i, q) = q^\top k_i$	Luong, Pham, and Manning
Scaled Dot Product	$a(k_i, q) = \frac{q^\top k_i}{\sqrt{d_k}}$	Vaswani, Shazeer, Parmar, Uszkor
General	$a(k_i, q) = q^\top W k_i$	Luong, Pham, and Manning
Biased General	$a(k_i, q) = k_i(W q + b)$	Sordoni, Bachman, and Bengio
Activated General	$a(k_i, q) = \text{act}(q^\top W k_i + b)$	Ma, Li, Zhang, and Wang
Generalized Kernel	$a(k_i, q) = \phi(q)^\top \phi(k_i)$	Choromanski, Likhoshesterov, Doha
Additive\Concat <sup>2</sup>	$a(k_i, q) = v^\top \text{act}(W[q; k_i] + b)$	Bahdanau, Cho, and Bengio, Luon
Deep	$a(k_i, q) = v^\top E^{(L-1)} + b^L$ $E(l) = \text{act}(W_l E^{(l-1)} + b^l)$ $E(1) = \text{act}(W_0 k_i + W_1 q + b^1)$	Pavlopoulos, Malakasiotis, and An
Location-based	$a(k_i, q) = \text{act}(W q)$	Luong, Pham, and Manning
Feature-based	$a(k_i, q) = v^\top \text{act}(W_0 \phi(K) + W_1 \phi(K) + b)$	Li, Kaiser, Bengio, and Si

Cuadro 2.1: Distintos tipos de funciones de alineación. (Tabla basada en [4] [39]).

$a(k_i, q)$  representa la función de alineación entre  $k_i$  y  $q$  y  $\text{act}$  es una función de activación.  $\text{sim}$  es una función de similaridad, Graves, Wayne, and Danihelka propone la función coseno. Los parámetros  $v, W, W_1, W_2$  son parámetros aprendidos por la red neuronal.

<sup>1</sup> El factor de escala  $\frac{1}{\sqrt{d_k}}$  ayuda a estabilizar cuando el gradiente es muy pequeño.  $d_k$  es el tamaño de la cabeza de atención.

<sup>2</sup> La función de activación propuesta por Bahdanau, Cho, and Bengio es la función  $\tanh$  como se ve en 2.21

secuencias de entrada y salida respectivamente, como es el caso del modelo propuesto por Bahdanau, Cho, and Bengio. El segundo tipo, **Co-Atención** (*co-attention*) utiliza distintas secuencias al mismo tiempo para conocer los pesos de atención entre estas entradas. Por ejemplo, en tareas en donde se necesita trabajar con datos multi-modales como procesar imágenes y texto simultáneamente. En tareas como *Visual Question Answering* se puede

aplicar un mecanismo de atención conjunto tanto para las imágenes y el texto para identificar las regiones de la imagen y las palabras del texto que son más relevantes. El tercer tipo es **auto-atención** (*Self Attention*), fue propuesto por Yang, Yang, Dyer, He, Smola, and Hovy y es uno de los puntos claves para los modelos *Transformers* [36]. Es comúnmente usada en tareas que solo requieren una salida resumen y no una secuencia como *Clasificación de texto*. La clave (key) y valor (value) pasan a ser las mismas y la atención es calculada sobre los mismos elementos pertenecientes a la secuencia de entrada, buscando así, encontrar las relaciones entre las palabras de la misma oración.

La segunda Categoría agrupa la atención por el nivel de abstracción en la que es aplicada, a un **solo nivel** o en **múltiples niveles**. La información a procesar muchas veces puede ser representada en distintos niveles de abstracción, es decir, en texto, podemos separar los datos a nivel de letras, n-gramas, palabras, oraciones, párrafos, etc., por tanto, es posible atender de manera jerárquica a las palabras que forman una oración para posteriormente prestar atención a las oraciones que conformar un texto más largo. Yang, Yang, Dyer, He, Smola, and Hovy utiliza este procedimiento para generar un vector de características usado posteriormente en un etapa de clasificación.

En la tercer categoría la atención es realizada en diversas partes de la secuencia; la suma pesada sobre todos los puntajes de las entradas usada por Bahdanau, Cho, and Bengio se le denomina **Atención suave** (*Soft-Attention*). Una alternativa es la **Atención dura** (*Hard-Attention*) [43] que calcula la atención no sobre todas los puntajes de alineamiento sino en una parte de estos, para ello se usa una distribución multinoulli parametrizada por los pesos de la atención. A pesar de que es más eficiente que la *atención suave* resulta difícil de entrenar al no ser completamente diferenciable. Otra opción a la *atención dura* es la **Atención Local** (*Local Attention*) cuya idea es aplicar atención sobre una ventana elegida ya sea centrada con respecto a la tiempo actual (alineamiento monótonico) o predicha por una función (alineamiento predictivo). La *atención local* fue propuesta por Luong, Pham, and Manning así como la *Atención Global* la cual es similar a la *atención suave*.

La última categoría divide los modelos de atención por las formas de representación de las entradas sobre las que la atención es aplicada. Distintos modelos pueden beneficiarse de procesar los datos creando vectores de características distintos, cada uno de ellos deriva de algún tipo de representación de la entrada. por tanto, es posible atender a diferentes representaciones y formar un vector final usando una combinación pesada de estos a través de dichos pesos de atención. Chaudhari, Polatkan, Ramanath, and Mithal llama a este tipo de modelos de atención como **multi-representational AM**. En la segunda categoría, **multi-dimensional attention**, la atención no es aplicada sobre los diversos vectores de características sino a un nivel más interno, sobre sus dimensiones. Pesando cada característica de un vector de características permite seleccionar aquellas que mejor lo describen para un contexto dado. EN *NLP*, resulta bastante útil cuando se trata con *polisemia*, en donde una palabra o frase puede tener más de un significado.

## 2.2. El modelo Transformer

A finales del año 2017 se presentó un nuevo modelo que vino a revolucionar el área de Procesamiento de Lenguaje Natural, El Transformer [37]. Una de sus principales características es la capacidad de procesar la información de alguna secuencia de forma paralela, caso contrario a las Redes Neuronales Recurrentes, donde la información se procesa recurrentemente. Gracias a ello, la capacidad de *recuerdo* no se ve afectado por el problema de *El desvanecimiento del Gradiente* específicamente cuando el problema es trabajar con secuencias bastante largas.

El *Transformer* puede ser visto como otro modelo *seq2seq* (Secuencia a Secuencia) 2.9, formado en por dos etapas, la primera encargada de codificar la información de entrada y la segunda de decodificarla, pero su principal característica es que aplica mecanismos de *Self-Attention* para capturar las dependencias globales entre la entrada y la salida. Dada una secuencia de entrada  $X = (x_1, x_2, \dots, x_n)$  con  $n$  como el tamaño de la secuencia, el codificador produce una representación intermedia  $Z = (z_1, z_2, \dots, z_n)$  al igual que los modelos *seq2seq*. El decodificador usa la secuencia  $Z$  para generar la secuencia de salida  $Y = (y_1, y_2, \dots, y_m)$  uno a la vez (en modo inferencia), con  $m$  como el tamaño de la secuencia de salida. Nótese, que el generar una salida a la vez el decodificador tiene que ser auto-regresivo. Usa la salida anterior  $y_{i-1}$  como entrada adicional para generar la siguiente salida  $y_i$ . Por ello, durante entrenamiento el modelo es alimentado con entradas y salidas desfasadas en un tiempo.

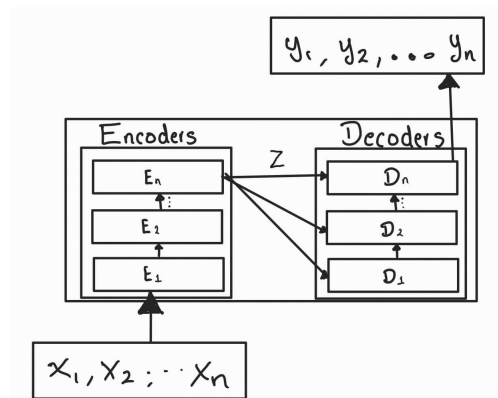
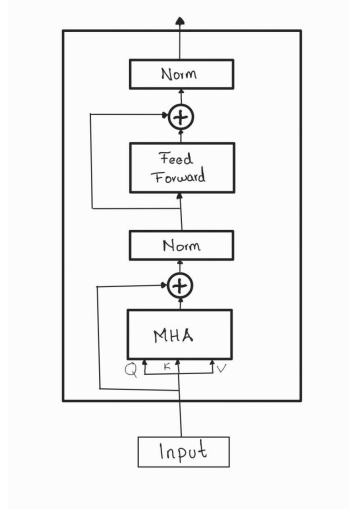


Figura 2.9: Modelo Transformer generalizado como modelo Secuencia a Secuencia

### 2.2.1. El Codificador y Decodificador

El *Modelo Transformer* está formado por múltiples codificadores y decodificadores apilados e inter-conectados, Como observamos en la figura 2.9. El codificador consta de dos capas, la primera de ellas aplica *Self-Attention* múltiples veces sobre la misma entrada (*Multi-*

*HeadSelf Attention*) y la segunda capa representada solo por una red *Feed-Forward* cuya entrada es la salida de la capa anterior. Véase la figura 2.10.



$$\begin{aligned}
 mha &= MHA(X, X, X) \\
 norm &= Norm(mha + X) \\
 f &= FeedForward(norm) \\
 Encoder(X) &= Norm(f + norm)
 \end{aligned}$$

Figura 2.10: Etapa Codificadora del Modelo Transformer. Pseudocódigo

El decodificador tiene una estructura similar al codificador con una etapa adicional intermedia de *Multi-Head Attention* aplicada sobre la salida de la pila de codificadores. También, la primer capa de atención sufre un ligero cambio en su forma de operación, necesitando enmáscarar (al momento en que se realiza el entrenamiento) la atención prestada del pasado al futuro. Esto es debido a que el decodificador se encarga de generar una secuencia (en modo inferencia) uno a la vez usando solamente la salida anterior y por tanto no tiene conocimiento de salidas futuras, observe la figura 2.12.

### 2.2.2. Multi-Head Self-Attention

En la sección 2.1.3 se detalla una generalización de la atención y diversas variantes usadas a lo largo de la literatura. El modelo original que introdujo a los Transformers usa en especial la variante *Scaled Dot-Product Attention*[37]:

$$Attention(q, k, v) = softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)v \quad (2.23)$$

El Transformer está basado en la idea de de aplicar atención multiples veces, al usar varias cabezas de atención, Multihead-Self-Attention (MHA), permite al modelo conjuntamente atender a información en distintas posiciones desde  $h$  diferentes subespacios de representación. 2.24



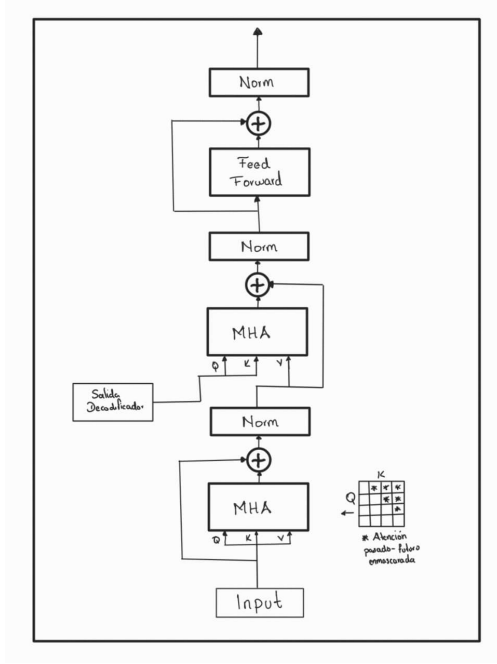


Figura 2.11: Etapa Decodificadora del Modelo Transformer. Pseudocódigo

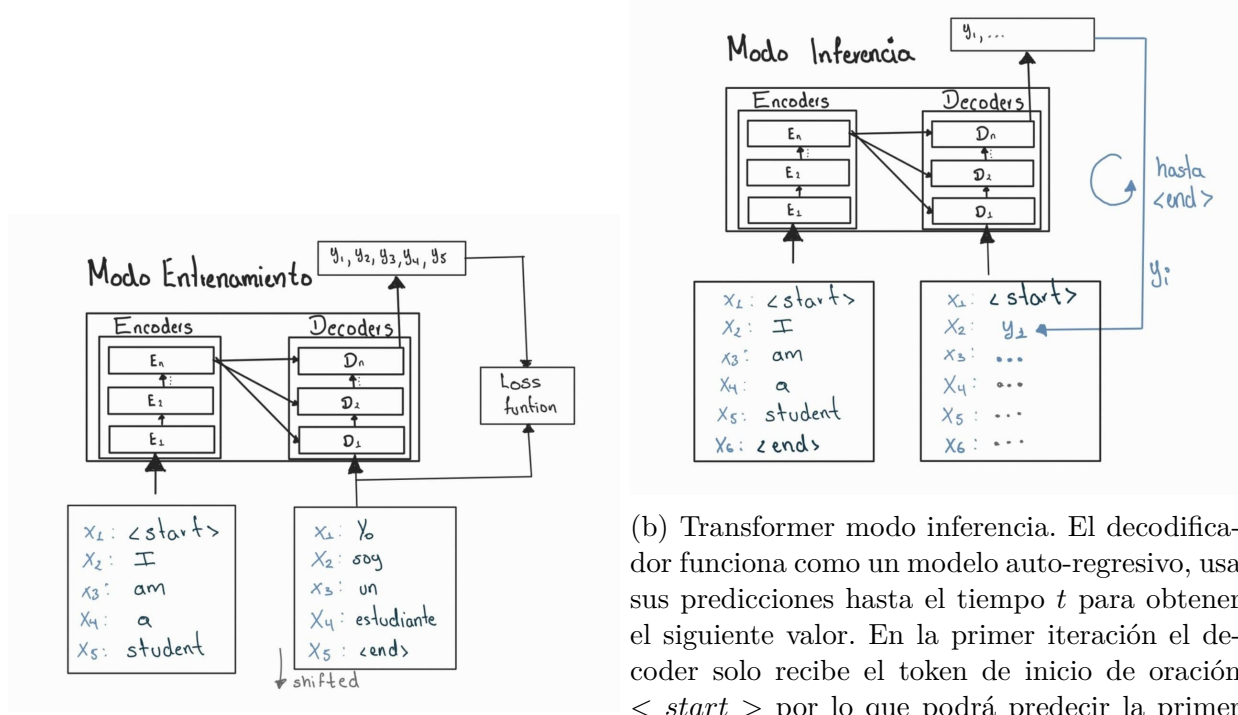
$$\begin{aligned}
 mha_1 &= MHA(X, X, X) \\
 norm_1 &= Norm(mha_1 + X) \\
 mha_2 &= MHA(enc_{out}, enc_{out}, norm_1) \\
 norm_2 &= Norm(mha_2 + X) \\
 f &= FeedForward(norm_2) \\
 decoder(X) &= Norm(f + norm_2)
 \end{aligned}$$

$$mha(Q, K, V) = Concat(head_1, head_2, head_3, \dots, head_h)W^O \quad (2.24)$$

Todas las cabezas de atención son concatenadas y resumidas para ser devueltas a las dimensiones del espacio de entrada original, principalmente para mantener consistencia en las dimensiones de usadas en cada etapa de codificación y decodificación del modelo a través de  $W^O \in \mathbb{R}^{hd_v \times d_m}$ .  $W^O$  es entrenado conjuntamente para aprender a resumir la información capturada por cada cabeza de atención.  $Q, K \in \mathbb{R}^{n \times d_m}$  y  $V \in \mathbb{R}^{n \times d_v}$  es la representación consulta, clave y valor de los embeddings de entrada de cada capa de atención del codificador y decodificador como se observa en las figuras 2.10 2.11.  $n$  es el tamaño de la secuencia,  $d_m$  y  $d_v$  son los tamaño del embedding y  $h$  el número de cabezas de atención.

En el caso del modelo transformer tenemos un conjunto embeddings sobre las cuales se aplica atención, si bien, no representan necesariamente las consultas, llaves, y valores utilizados para la atención generalizada, podemos obtener estas representaciones transportándolos a sus espacios respectivos a través de alguna transformación aprendida conjuntamente con el entrenamiento del modelo.

Por tanto, para el conjunto de Embeddings  $E_Q \in \mathbb{R}^{n \times d_m}$ ,  $E_K \in \mathbb{R}^{n \times d_m}$  y  $E_V \in \mathbb{R}^{n \times d_v}$  donde  $n$  es el número embeddings,  $d_m$  y  $d_v$  son las dimensiones de cada uno, la atención en cada cabeza  $i$  se calcula como:



(a) Transformer modo entrenamiento. Las entradas en el decodificador son recorridas un elemento en el futuro, con el fin de que aprende a predecir la siguiente palabra dado un contexto previo agregado como entrada al decodificador. El decodificador termina su predicción en el momento que el token  $\langle \text{end} \rangle$  es obtenido.

Figura 2.12: Esquema de entrenamiento e inferencia del modelo Transformer en un problema de Machine Translation.

$$\begin{aligned} Q_i &= E_Q W_i^Q \\ K_i &= E_K W_i^K \\ V_i &= E_V W_i^V \end{aligned} \quad (2.25)$$

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \quad (2.26)$$

donde  $W_i^Q, W_i^K \in \mathbb{R}^{d_m \times d_k}$ ,  $W_i^V \in \mathbb{R}^{d_m \times d_v}$  y  $d_k = d_v = d_m/h$ .

el término de escalamiento  $\sqrt{d_k}$  ayuda a evitar que la magnitud de los productos puntos

calculados entre cada consulta y llave crezcan demasiado, y que la función *softmax* pueda ser más estable al evitar regiones donde los gradientes son muy pequeños[37].

### 2.2.3. Información Posicional

En los modelos basados en *Redes Recurrentes* la información se procesan uno a uno en cada paso de tiempo. Los modelos basados en *Transformers* procesan la información en conjunto, perdiendo la noción de la temporalidad de los datos. Una solución es agregar dicha información perdida a través de vectores que codifiquen el tiempo/posición de los datos sumándolos con los vectores de embeddings. Estos vectores llamados *Positional Encodings* [12] siguen un patrón en específico que el modelo aprende a identificar y lo ayuda a determinar la posición de cada elemento de la secuencia y por tanto calcular a qué distancia se encuentra cada uno de los demás.

Por lo regular se usa una onda senoidal y cosenoidal para lugares pares e impares, formando una progresión geométrica desde  $2\pi$  hasta  $10000 \cdot 2\pi$  2.27:

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_m}) \\ PE(pos, 2i + 1) &= \cos(pos/10000^{2i/d_m}) \end{aligned} \quad (2.27)$$

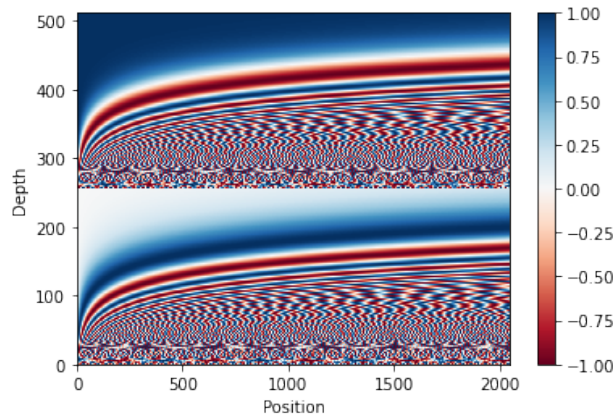


Figura 2.13: 2000 Vectores de Positional Encoding con dimensiones de embedding=500.

Poner una figura completa de todo el esquema del Transformer

### 2.2.4. Problemas típicos en el entrenamiento de Transformers

#### Learning Rate WarmUp y Layer Normalization

A pesar de que la arquitectura del modelo Transformer no es tan compleja, puesto que tanto el codificador como el decodificador están formados por pilas de capas de atención y MLP, el entrenamiento de este tipo de modelos muchas veces no resulta tan trivial. Regularmente requiere de una combinación de técnicas para lograr su convergencia a valores aceptables y en conjunto con una gran cantidad de datos, tamaños de lotes de procesamiento grandes y una gran cantidad de tiempo de procesamiento en gpu [30].

*Learning Rate WarmUp* es una de las primeras técnicas usadas y descritas en el proceso de entrenamiento por Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin. Usando el algoritmo Adam como optimizador se varía el factor de aprendizaje de acuerdo a la fórmula:

$$lrate = d_m^{-0,5} \cdot \min \left( step\_num^{-0,5}, step\_num \cdot warmup\_steps^{-0,5} \right) \quad (2.28)$$

En el esquema anterior, más conocido como *Noam-Warmup*, el modelo original es entrenado incrementando linealmente el factor de aprendizaje en los primeros  $warmup\_steps = 4000$  pasos. Posteriormente, decreta propocionalmente al inverso del la raíz cuadrada del paso  $step\_num$  actual, véase la figura 2.14.

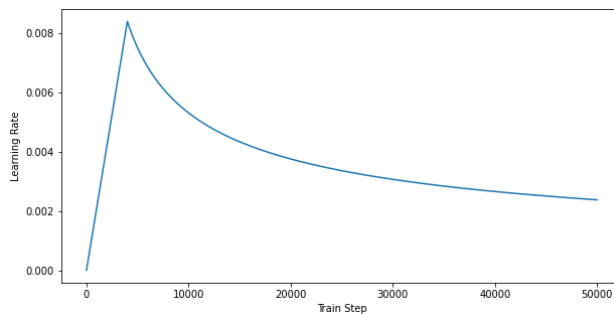


Figura 2.14: Noam-Warmup con  $warmup\_steps = 4000$  y  $d_m = 512$

Si bien, la razón por la que funciona este tipo de técnica no está del todo claro, se presume que usar *Learning Rate WarmUp* ayuda a reducir la varianza del factor de aprendizaje adaptativo durante las primeras etapas del entrenamiento del modelo. Liu, Jiang, He, Chen, Liu, Gao, and Han demostraron que el segundo momento del algoritmo de Adam durante etapas tempranas de optimización es proporcional a una integral divergente, lo que provoca las actualizaciones inestables, llevando al modelo fuera de las regiones donde un mejor mínimo existe. Con esto en mente Liu, Jiang, He, Chen, Liu, Gao, and Han proponen el algoritmo de

optimización *RAdam* (Rectified Adam) como una alternativa a usar *Learning Rate WarmUp* y mitigar este efecto durante la fase inicial del entrenamiento de los modelos.

El *Learning Rate WarmUp* comúnmente es usado en conjunto con algoritmos de optimización estocásticos como *RMSprop* o *Adam*. En vez configurar el *factor de aprendizaje*  $\alpha$  con un decremento constante, la estrategia de *Learning Rate WarmUp* configura este factor con valores muy pequeños en los primeros pasos de entrenamiento. Durante las primeras etapas del entrenamiento el factor de aprendizaje es incrementado hasta un límite que es ligeramente superior o inferior al valor inicial de  $\alpha$  del optimizador usado y posteriormente es decrementado progresivamente hasta la convergencia del modelo.

Así, en cada paso del algoritmo de optimización el cuál está parametrizado por el factor de aprendizaje  $\alpha$ , puede ser aplicado un factor de *warmup*  $\omega \in [0, 1]$  que sirve para reducir  $\alpha$  y a la vez el paso de optimización en cada tiempo, replazando  $\alpha_t = \alpha\omega_t$ . La forma mas sencilla es usar un factor **linear warmup** parametrizado por un periodo de “calentamiento”  $\tau$ .

$$\omega_t^{linear, \tau} = \min\left(1, \frac{t}{\tau}\right) \quad (2.29)$$

Ma and Yarats proponen 3 formas de aplicar la técnica de *warmup*:

**Exponential warmup** aplica un decaimiento exponencial

$$\omega_t^{expo, \tau} = 1 - \exp\left(-\frac{1}{\tau}t\right) \quad (2.30)$$

recomienda elegir  $\tau = (1 - \beta_2)^{-1}$  tal que no se tan diferente del segundo momento de corrección de bias del algoritmo de *Adam*  $\beta_2$ .

$$\omega_t^{expo, untuned} = 1 - \exp(-(1 - \beta_2)t) \quad (2.31)$$

Similar al decaimiento exponencial proponen usar *linear warmup* sobre  $\tau = 2(1 - \beta_2)^{-1}$  iteraciones para preservar un efecto similar de des-aceleración con el paso del tiempo.

$$\omega_t^{linear, untuned} = \min\left(1, \frac{1 - \beta_2}{2}t\right) \quad (2.32)$$

Por otro lado, Huang, Perez, Ba, and Volkovs mencionan que usar la técnica de *Learning Rate WarmUp* para mitigar la varianza del optimizador Adam no es del todo la solución y que el problema radica precisamente en la arquitectura del modelo Transformer, principalmente en las capas de normalización [5] [42]. En particular Xiong, Yang, He, Zheng, Zheng,

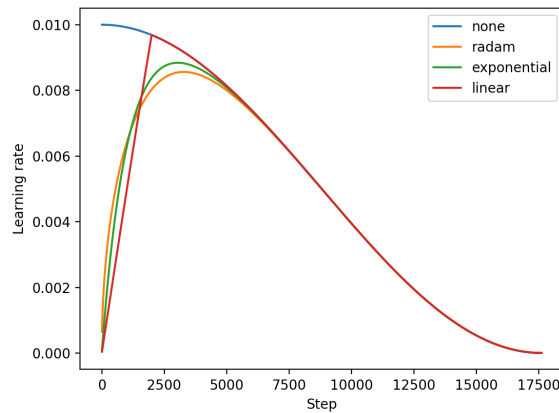


Figura 2.15: Learning rate sobre X 18000 iteraciones usando RAdam y lineal, exponencial warmup con Adam

Xing, Zhang, Lan, Wang, and Liu encuentran que para un modelo Transformer de cualquier tamaño con capas de normalización entre bloques residuales (*Post-LN Transformer*), la escala de la norma del gradiente que incide en la última capa de normalización permanecen igual al no depender de la cantidad de bloques del transformer. Por el contrario, si la capa de normalización es colocada justo antes de la conexión residual (*Pre-LN Transformer*) la magnitud de la norma del gradiente decrece conforme el tamaño del modelo incrementa, guiando así, al problema de desvanecimiento de gradiente. Huang, Perez, Ba, and Volkovs proponen eliminar las capas de normalización del modelo Transformer que en conjunto con la inestabilidad del algoritmo de optimización de Adam provocan la dificultad de entrenamiento durante desde las primeras etapas. Para ello, estandarizan la siguiente inicialización (*T-Fixup*) de pesos del modelo, permitiendo evitar la etapa de *WarmUp* y las capas de normalización en el Transformer. La figura 2.16 muestra una comparativa de los histogramas usando la inicialización *T-Fixup* e usar el algoritmo de Adam con y sin etapa de *Warmup*:

- Aplicar initialization tipo *Xavier* para todos los pesos del modelo. Excepto el proceso de generación de embedding adecuados al tamaño del modelo  $d_m$ .
- Usar una inicialización tipo *Gaussiana* con  $\mathbb{N}(0, d_m^{\frac{1}{2}})$  para los pesos de generación de embeddings.
- Escalar las matrices  $W_i^V$  y  $W^O$  en cada bloque de atención en el decodificador, los pesos en de cada capa MLP del decodificador y los pesos de generación de embeddings tanto del codificador como decodificador por  $9N^{-\frac{1}{4}}$  donde  $N$  es el número de bloques del Transformer.
- Escalar las matrices  $W_i^V$  y  $W^O$  de cada bloque de atención del codificador y los pesos de cada capa de MLP del codificador por  $0,67N^{-\frac{1}{4}}$

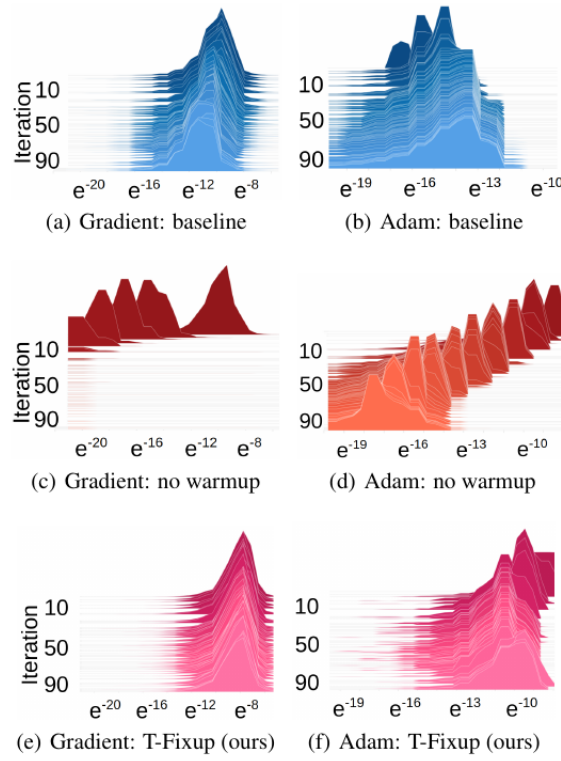


Figura 2.16: Histograma de gradientes del Algoritmo Adam con y sin etapa de *WarmUp* y usando inicialización *T-Fixup*. Imagen original de Huang, Perez, Ba, and Volkovs

## Cálculo de la Atención

Además de lo específico y delicado del entrenamiento del Modelo Transformer su costo en tiempo computacional y de memoria también representa un serio problema a la hora de optimizar e inferir. Debido principalmente a que en el proceso de atención debe focalizar cada token con respecto a todos los demás, lo que lleva a que su complejidad crezca cuadráticamente con respecto a el tamaño de la secuencia.

Varías técnicas han sido propuestas para reducir este problema, muchas de ellas involucran en reducir la atención a vecindades de representaciones, aproximar la matriz de atención con otras matrices de transformaciones a través de kernels o sustituir completamente la operación *softmax* por otra función.

### Atención de vecindades:

Parmar, Vaswani, Uszkoreit, Kaiser, Shazeer, and Ku particionan la información de las representaciones de las consultas asignándolos a diferentes bloques de memoria, restringiéndose a vecindarios locales alrededor de cada consulta. Principalmente basados en cómo las redes

convolucionales trabajar. Sin embargo esta solución es parcial y solo aplicable a secuencias de datos con relaciones cortas, cómo imágenes.

Child, Gray, Radford, and Sutskever factorizan la matriz de atención para reducir su complejidad de  $O(n^2)$  a  $O(n\sqrt{n})$  por medio de matrices ralas, separando la atención a través de diferentes pasos al parametrizar la atención por una conectividad de distintos patrones elegidos previamente bajo el supuesto de que las matrices de atención son ralas, puesto que no contienen dependencias de relevancia sobre representaciones distantes como se observa en la imagen 2.17.

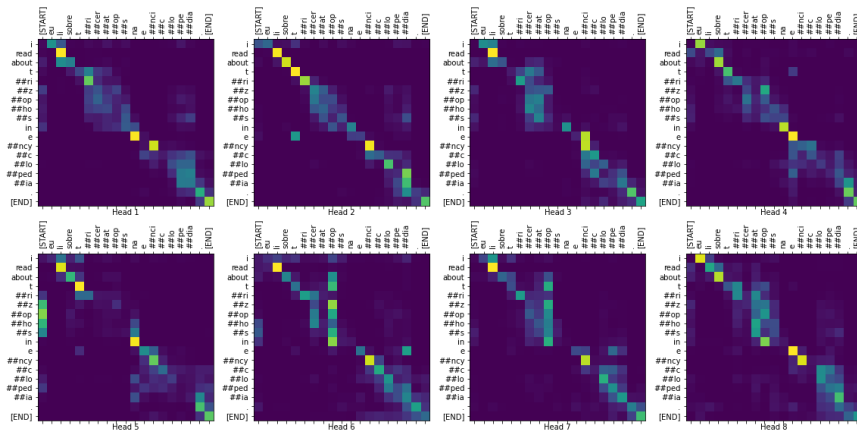


Figura 2.17: Visualización de 8 cabezas de atención sobre una tarea de Machine-Translation. Las matrices de atención tienden a ser ralas al tener carencia de relaciones relevantes entre diversas representaciones a distancias lejanas.

Sukhbaatar, Grave, Bojanowski, and Joulin proponen reducir el ancho de la atención basándose en que cada representación no necesita prestar atención sobre todas las demás sino que debería ser adaptativo. Así, para cada cabeza de atención se agrega una función de enmascaramiento que controla la flexibilidad del ancho una ventana. La ventana formada cambia dinámicamente de tamaño dependiendo de la representación en cuestión. Beltagy, Peters, and Cohan siguen una estrategia similar, implementando atención local con ventanas dilatadas distintas para cada cabeza, permitiendo atender contextos menos locales en cada ocasión y atención global sobre preseleccionadas localizaciones. Dada la dificultad de su implementación sin usar ciclos para iterar sobre los elementos seleccionados a atender, implementan su propio kernel en *CUDA* con las operaciones optimizadas para realizar esta tarea.

Dai, Yang, Yang, Carbonell, Le, and Salakhutdinov mencionan que si el problema es el procesamiento de grandes secuencias por qué no dividir las en secuencias más pequeñas y procesarlas individualmente y así evitar usar grandes cantidades de memoria en su procesamiento. El principal problema de este enfoque es que cada secuencia es procesada individualmente y la información de previas secuencias es ignorada evitando que esta fluya a través



de las próximas secuencias. Para solucionar este inconveniente introducen un mecanismo de recurrencia en la arquitectura del transformer. Durante el entrenamiento (véase la figura 2.18) un estado oculto es calculado de las secuencias previas y guardado en memoria para extender el contexto al momento de procesar la siguiente secuencia. Durante el proceso de evaluación, el resultado de las operaciones del transformer pueden ser reusado y no calculado nuevamente desde cero, permitiendo reducir considerablemente el tiempo de evaluación.

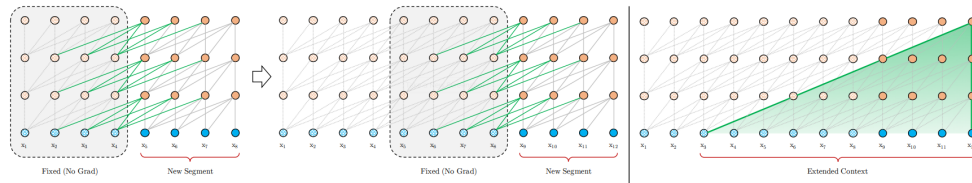


Figura 2.18: Transformer-XL. Para tratar con secuencias largas divide el proceso en secuencias más cortas creando estados ocultos intermedios y usándolos en el cálculo de las próximas secuencias. Figura obtenida de [11].

Kitaev, Kaiser, and Levskaya reducen el problema de realizar la operación de softmax sobre toda la matriz  $Q_i K_i^T$  a calcularlo individualmente por cada consulta  $q_j$ , guardando solo una vez en memoria este valor en cada iteración y recalculándolo cuando se necesite de nuevo al utilizar *Back-Propagation* usando de capas reversibles. Si bien, computacionalmente es más costoso permite usar mucho menos memoria que la solución original. Por otro lado, dado que el resultado de la función softmax depende en mucho mayor medida en los elementos dominantes de la matriz, solo es necesario fijarse en las llaves más cercanas a la consulta en cuestión. *LSH (Local Sensitive Hashing)* resuelve este problema permitiendo encontrar rápidamente los vecinos más cercanos en espacios de altas dimensiones, con la restricción de que  $W_i^Q = W_i^K$  dado que se necesita conservar la similitud entre consultas y llaves, algo que sería más difícil si sus matrices de proyección  $W_i^Q$  y  $W_i^K$  fuesen muy distintas.

### Aproximaciones a la Atención original:

También Xiao, Li, Zhu, Yu, and Liu proponen un modelo para compartir pesos de capas adyacentes (Shared Attention Network - SAN). Cada  $\pi$  capas continuas en el codificador comparten la misma matriz de atención y en el decodificador se comparte la proyección de los pesos de atención sobre la representación de los valores  $V_i$ , en otras palabras se comparte directamente la cabeza de atención  $head_i$ . Dado que no es tan fácil conocer que capas deben compartir pesos, establecen un proceso iterativo de entrenamiento basados en calcular que tan diferentes son las capas del transformer usando la *Divergencia de Jensen-Shannon*. Si la similitud entre dos capas es mayor a cierto umbral se indica que dichas capas deben compartir pesos. Se repite un nuevo entrenamiento y se calcula nuevamente la similitud entre capas, y así sucesivamente hasta convergencia. Podemos notar que este proceso de entrenamiento y ajuste de pesos es muy costoso, un nuevo entrenamiento es requerido por cada ajuste de compartición de pesos, pero el modelo resultante es menos complejo y el

tiempo en modo de evaluación o inferencia se ve reducido considerablemente.

Wang, Li, Khabsa, Fang, and Ma bajo la hipótesis de que la matriz de atención tienen rango mucho menor que  $n$  proponen obtener los valores de cada cabeza haciendo una aproximación a ella. Para ello, se hace uso de dos matrices entrenables conjuntamente con el modelo,  $E$  y  $F \in \mathbb{R}^{n \times k}$  con  $k \ll n$ , tal que,  $head_i = softmax(\frac{Q_i(E_i K_i)}{\sqrt{d_m}}) F_i V_i$ . Con ello la dimensión correspondiente al tamaño de las secuencias se ve reducido bajo el supuesto que podemos representar la información secuencial en un espacio más pequeño sin gran pérdida de información.

Choromanski, Likhoshesterov, Dohan, Song, Gane, Sarlós, Hawkins, Davis, Mohiuddin, Kaiser, Belanger, Colwell, and Weller por el contrario descomponen la operación de atención sobre los valores  $head = softmax(\frac{QK^T}{\sqrt{d_k V}})$  en una multiplicación matricial más simple  $head = Q'k'^T V$  con  $Q'$  y  $k'^T \in \mathbb{R}^{n \times r}$  y  $r \leq n$ . Para ello construyen  $Q'$  y  $k'$  como dos matrices usando kernels tal que su producto forma una aproximación a la función softmax aplicada al producto de  $Q$  y  $K$ . Notemos que con ello podemos reducir el costo computacional y en memoria simplemente reduciendo el producto  $Q'(k'^T V)$  de derecha a izquierda.

Finalmente autores como Lee-Thorp, Ainslie, Eckstein, and Ontañón rempazan completamente el bloque de *Multihead Attention* con bloques que aplican operaciones de Transformada de Fourier o lo largo de la dimensión de los embeddings y de las secuencias. Probando que el usar *FFT* (Fast Fourier Transform) es suficiente para abstraer y modelar las relaciones. Y como Wu, Wu, Qi, Huang, and Xie que cambian la atención entre todas las consultas y llaves por una sola con todas las llaves. Para esto, a través de atención suman todos las consultas en una consulta global. Este proceso es repetido con las llaves y valores como se observa en la figura 2.19

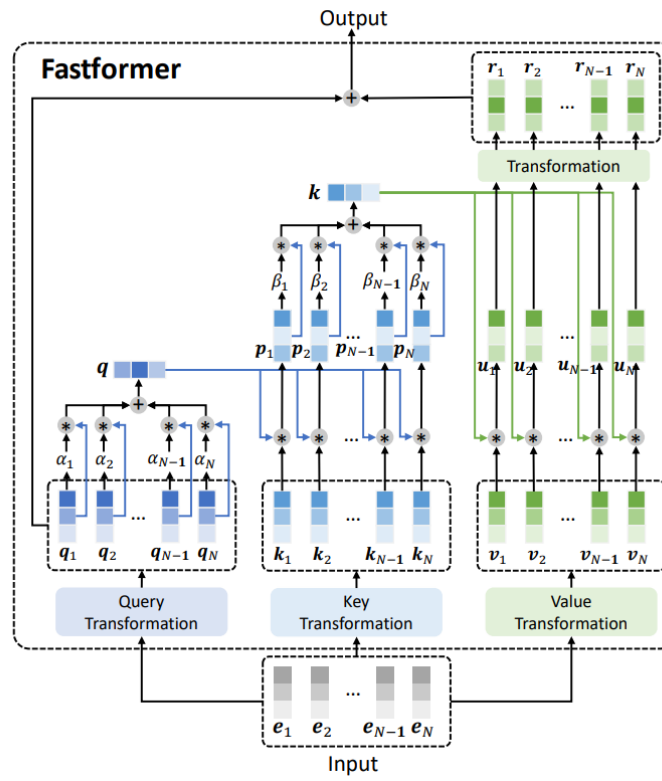


Figura 2.19: Fast-Former. Reemplaza la atención tradicional del transformer por una iterativa. En cada paso crea una consulta y clave global usando atención sobre estos mismos. Figura obtenida de [40]

# Apéndice A

## An appendix

Appendices are a good idea for almost any thesis. Your main thesis body will likely contain perhaps 40-60 pages of text and figures. You may well write a larger document than this, but chances are that some of the information contained therein, while important, does *not* merit a place in the main body of the document. This sort of content - peripheral clarifying details, computer code, information of use to future students but not critical to understanding your work ... - should be allocated to one or several appendices.

### A.1. About the bibliography

What follows this is the bibliography. This has its own separate environment and syntax; check out the comments in the .tex files for details. Worth nothing, though, is that you may find it helpful to use automated bibliography management tools. BibTeX will automatically generate a bibliography from you if you create a database of references. Other software - for example JabRef on a pc - can be used to make managing the reference database easy. Regardless, once you've created a .bib file you can cite it in the body of your thesis using the `\cite` tag. For example, one might wish to cite a reference by Bermudez [? ]. If you use BibTeX, you can put the relevant information into a referencedatabase (called bibliography.bib here), and then BibTeX will compile the references into a .bbl file ordered appropriately for your thesis based on when the citations appear in the main document.

# Bibliografía

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.
- [2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020. URL <https://arxiv.org/abs/2004.05150>.
- [3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.
- [4] Sneha Chaudhari, Gungor Polatkan, Rohan Ramanath, and Varun Mithal. An attentive survey of attention models. *CoRR*, abs/1904.02874, 2019. URL <http://arxiv.org/abs/1904.02874>.
- [5] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George F. Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. The best of both worlds: Combining recent advances in neural machine translation. *CoRR*, abs/1804.09849, 2018. URL <http://arxiv.org/abs/1804.09849>.
- [6] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019. URL <http://arxiv.org/abs/1904.10509>.
- [7] KyungHyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL <http://arxiv.org/abs/1409.1259>.
- [8] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL <http://arxiv.org/abs/1406.1078>.

- [9] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. *CoRR*, abs/2009.14794, 2020. URL <https://arxiv.org/abs/2009.14794>.
- [10] Junyoung Chung, Çağlar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL <http://arxiv.org/abs/1412.3555>.
- [11] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019. URL <http://arxiv.org/abs/1901.02860>.
- [12] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017. URL <http://arxiv.org/abs/1705.03122>.
- [13] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2017. <http://www.deeplearningbook.org>.
- [14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL <http://arxiv.org/abs/1410.5401>.
- [15] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [16] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4475–4483. PMLR, 13–18 Jul 2020. URL <https://proceedings.mlr.press/v119/huang20f.html>.
- [17] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *CoRR*, abs/2001.04451, 2020. URL <https://arxiv.org/abs/2001.04451>.
- [18] John F. Kolen and Stefan C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. 2001. doi: 10.1109/9780470544037.ch14.
- [19] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontañón. Fnet: Mixing tokens with fourier transforms. *CoRR*, abs/2105.03824, 2021. URL <https://arxiv.org/abs/2105.03824>.
- [20] Yang Li, Lukasz Kaiser, Samy Bengio, and Si Si. Area attention. *CoRR*, abs/1810.10126, 2018. URL <http://arxiv.org/abs/1810.10126>.

- [21] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *CoRR*, abs/1908.03265, 2019. URL <http://arxiv.org/abs/1908.03265>.
- [22] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL <http://arxiv.org/abs/1508.04025>.
- [23] Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Interactive attention networks for aspect-level sentiment classification. *CoRR*, abs/1709.00893, 2017. URL <http://arxiv.org/abs/1709.00893>.
- [24] Jerry Ma and Denis Yarats. On the adequacy of untuned warmup for adaptive optimization. *CoRR*, abs/1910.04209, 2019. URL <http://arxiv.org/abs/1910.04209>.
- [25] André F. T. Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *CoRR*, abs/1602.02068, 2016. URL <http://arxiv.org/abs/1602.02068>.
- [26] André F. T. Martins, Marcos V. Treviso, António Farinhas, Vlad Niculae, Mário A. T. Figueiredo, and Pedro M. Q. Aguiar. Sparse and continuous attention mechanisms. *CoRR*, abs/2006.07214, 2020. URL <https://arxiv.org/abs/2006.07214>.
- [27] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. *CoRR*, abs/1802.05751, 2018. URL <http://arxiv.org/abs/1802.05751>.
- [28] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28(3) of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013.
- [29] John Pavlopoulos, Prodromos Malakasiotis, and Ion Androutsopoulos. Deeper attention to abusive user content moderation. pages 1125–1135, 01 2017. doi: 10.18653/v1/D17-1117.
- [30] Martin Popel and Ondrej Bojar. Training tips for the transformer model. *CoRR*, abs/1804.00247, 2018. URL <http://arxiv.org/abs/1804.00247>.
- [31] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL <https://doi.org/10.1038/323533a0>.
- [32] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi: 10.1109/78.650093.

- [33] Alessandro Sordoni, Philip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *CoRR*, abs/1606.02245, 2016. URL <http://arxiv.org/abs/1606.02245>.
- [34] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. *CoRR*, abs/1905.07799, 2019. URL <http://arxiv.org/abs/1905.07799>.
- [35] Yi Tay, Anh Tuan Luu, Aston Zhang, Shuohang Wang, and Siu Cheung Hui. Compositional de-attention networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/16fc18d787294ad5171100e33d05d4e2-Paper.pdf>.
- [36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL <http://arxiv.org/abs/1706.03762>.
- [37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL <http://arxiv.org/abs/1706.03762>.
- [38] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768, 2020. URL <https://arxiv.org/abs/2006.04768>.
- [39] Lilian Weng. Attention? attention! *lilianweng.github.io/lil-log*, 2018. URL <http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>.
- [40] Chuhan Wu, Fangzhao Wu, Tao Qi, Yongfeng Huang, and Xing Xie. Fastformer: Additive attention can be all you need. *CoRR*, abs/2108.09084, 2021. URL <https://arxiv.org/abs/2108.09084>.
- [41] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. Sharing attention weights for fast transformer. *CoRR*, abs/1906.11024, 2019. URL <http://arxiv.org/abs/1906.11024>.
- [42] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. *CoRR*, abs/2002.04745, 2020. URL <https://arxiv.org/abs/2002.04745>.
- [43] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015. URL <http://arxiv.org/abs/1502.03044>.



- [44] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *NAACL 2016*, pages 1480–1489, June 2016. URL <https://www.microsoft.com/en-us/research/publication/hierarchical-attention-networks-document-classification/>.