

Lecture
RNN y transformadores para modelado
secuencia a secuencia

RNN de varios a varios para generar texto

1. - **Generación de secuencias con RNN**
2. - Caracter RNN en PyTorch
3. - RNN con atención
4. - Atención es todo lo que necesitamos
5. - Modelos de transformadores
6. - Transformador en PyTorch

Diferentes tipos de tareas de modelado de secuencias

Anteriormente, creamos un clasificador RNN (a nivel de palabra)

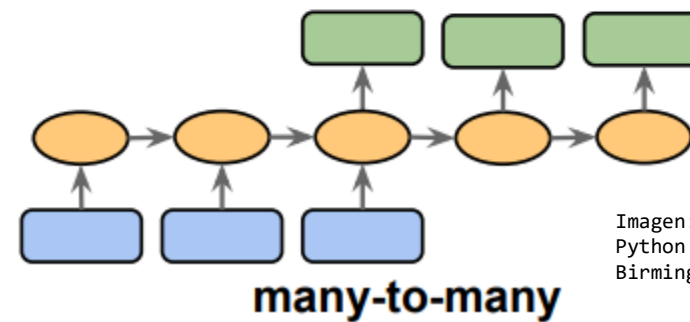
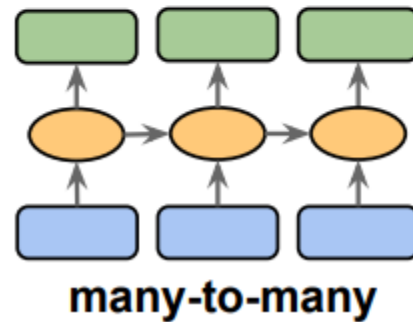
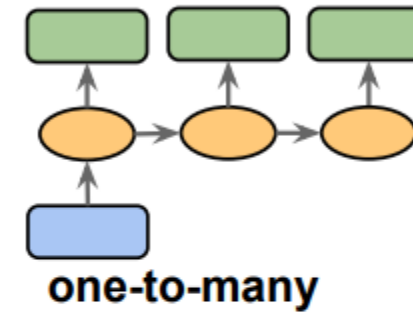
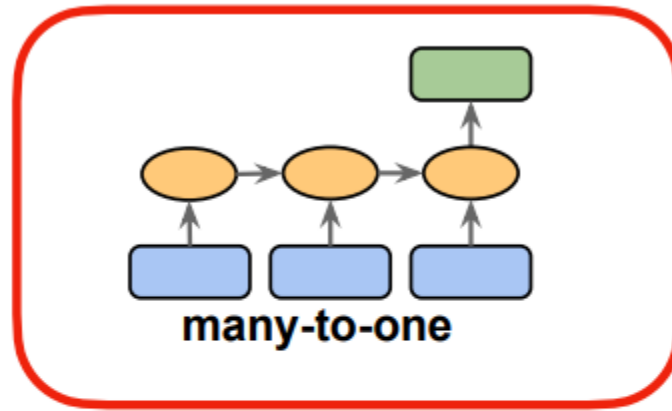
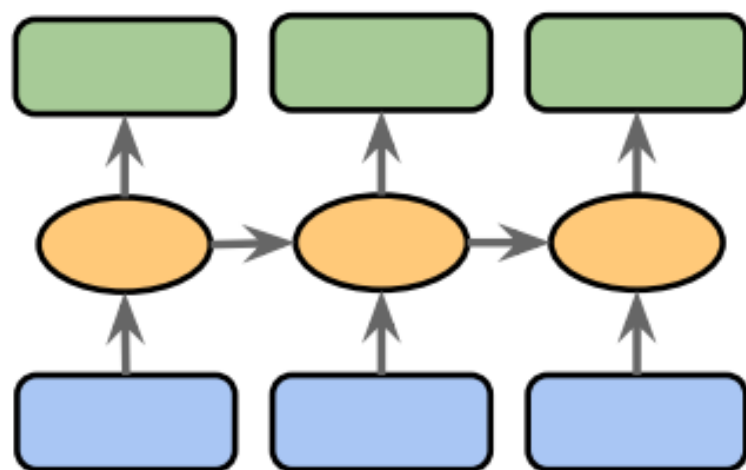
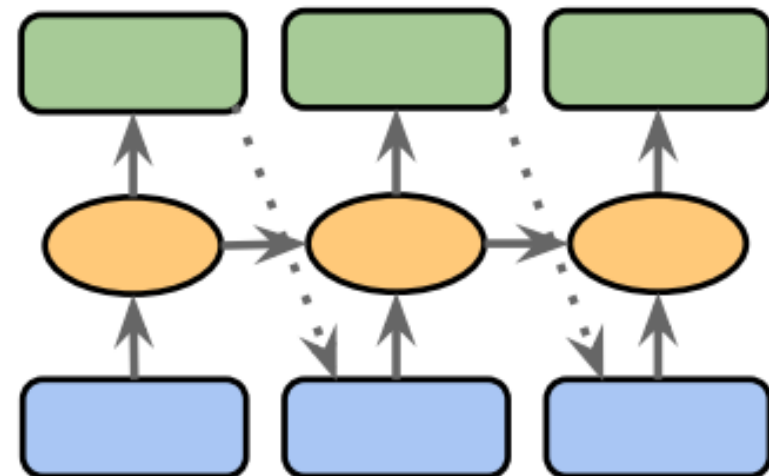


Imagen: Sebastian Raschka, Vahid Mirjalili.
Python Machine Learning. 3rd Edition.
Birmingham, UK: Packt Publishing, 2019



many-to-many

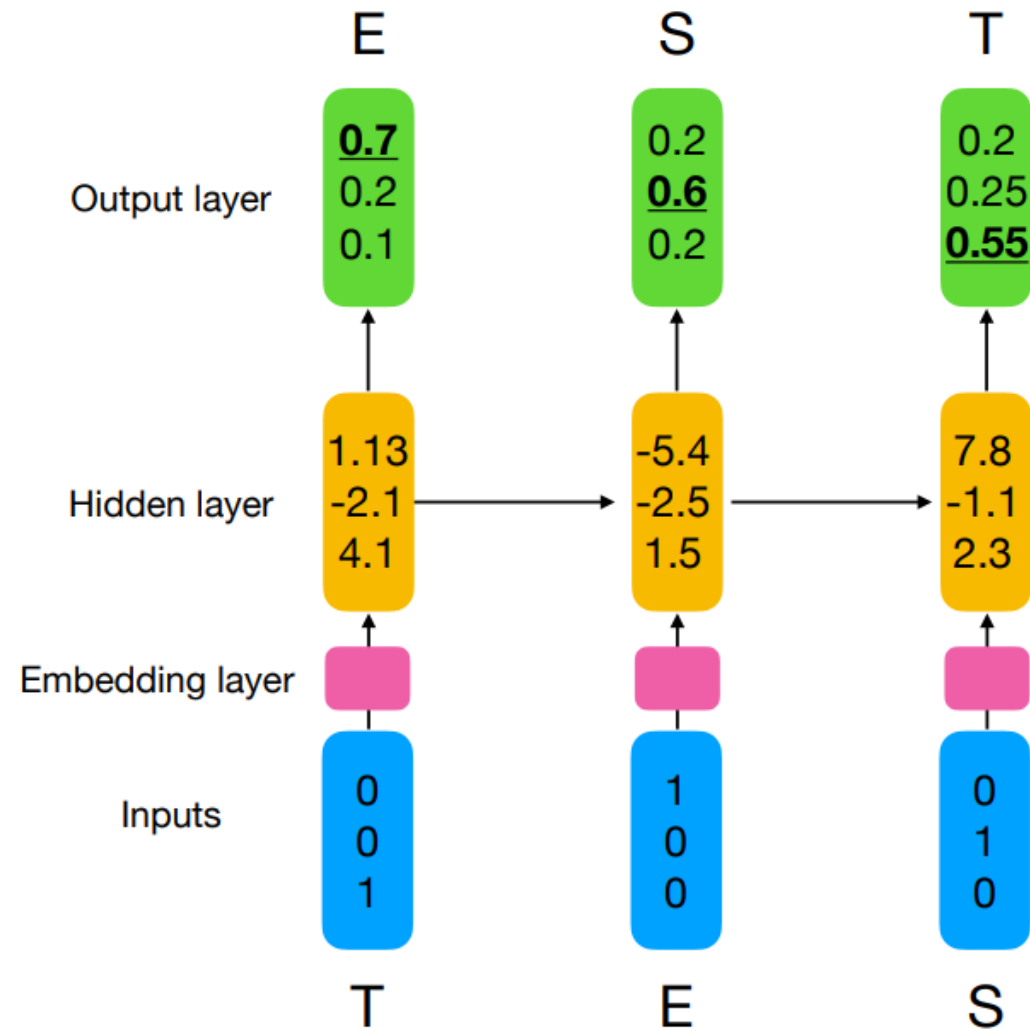
"training"



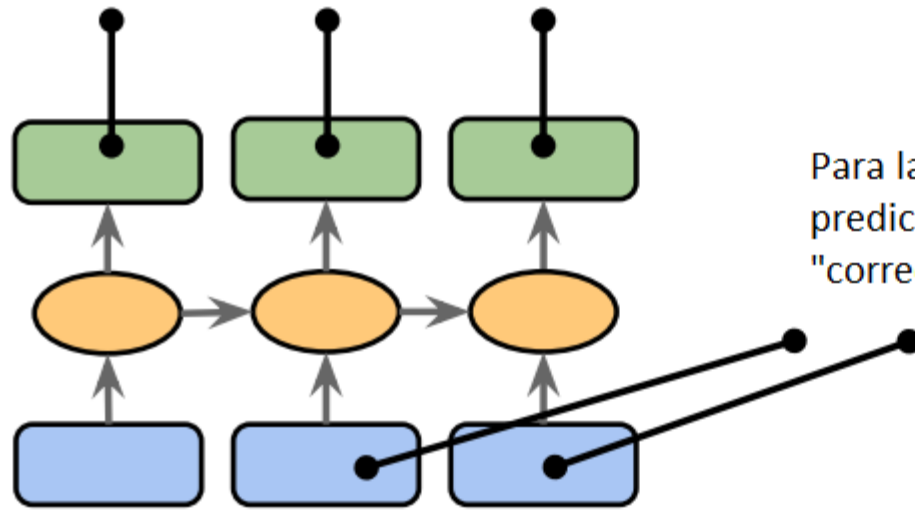
~~many-to-many~~
"one"

"generating new text"

Character RNN



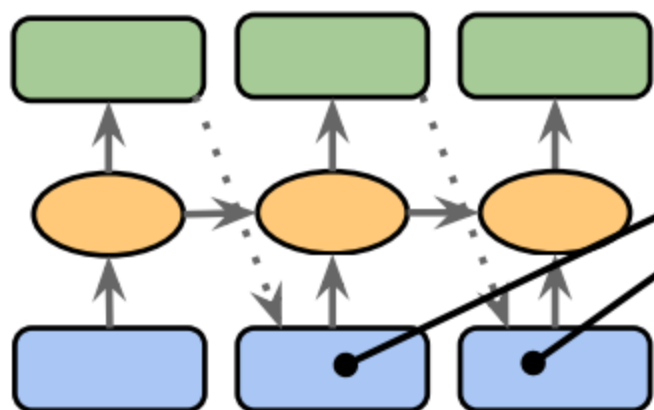
En cada paso de tiempo Salida
Softmax (probabilidad) para
cada posible "siguiente letra"



Para la siguiente entrada, ignore la
predicción pero use la siguiente letra
"correcta" del conjunto de datos

many-to-many

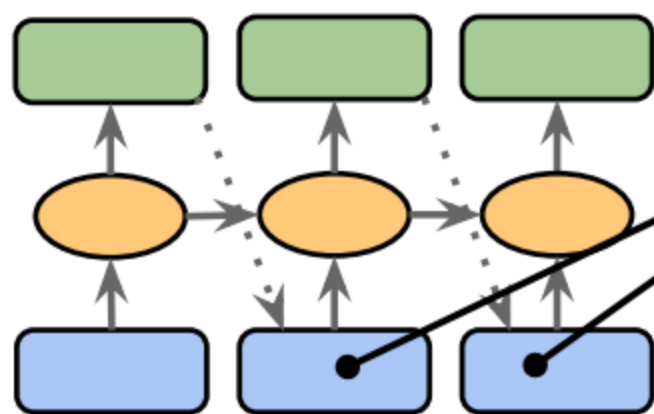
"training"



Para generar texto nuevo, ahora, muestre las salidas de softmax y proporcione la letra como entrada para el siguiente paso de tiempo

~~many-to-many~~
"one"

"generating new text"



Para generar texto nuevo, ahora, muestre las salidas de softmax y proporcione la letra como entrada para el siguiente paso de tiempo

~~many-to-many~~
"one"
"generating new text"

Tenga en cuenta que este enfoque funciona tanto con palabras como con caracteres RNN

Ventajas y desventajas de los RNN de caracter sobre los RNN de palabras

Ventajas:

- Las incrustaciones de caracteres (solo 24 letras más puntuación en inglés) requieren menos memoria en comparación con las incrustaciones de palabras
- Capas de salida más pequeñas por la misma razón que la anterior

Desventajas:

- Puede crear palabras extrañas y sin sentido
- Peor en la captura de dependencias de larga distancia

RNN de varios a varios para generar texto

1. - Generación de secuencias con RNN
2. - **Caracter RNN en PyTorch**
3. - RNN con atención
4. - Atención es todo lo que necesitamos
5. - Modelos de transformadores
6. - Transformador en PyTorch

Clase LSTM

<https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html>

Parameters

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights b_{ih} and b_{hh} . Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If > 0 , will use LSTM with projections of corresponding size. Default: 0

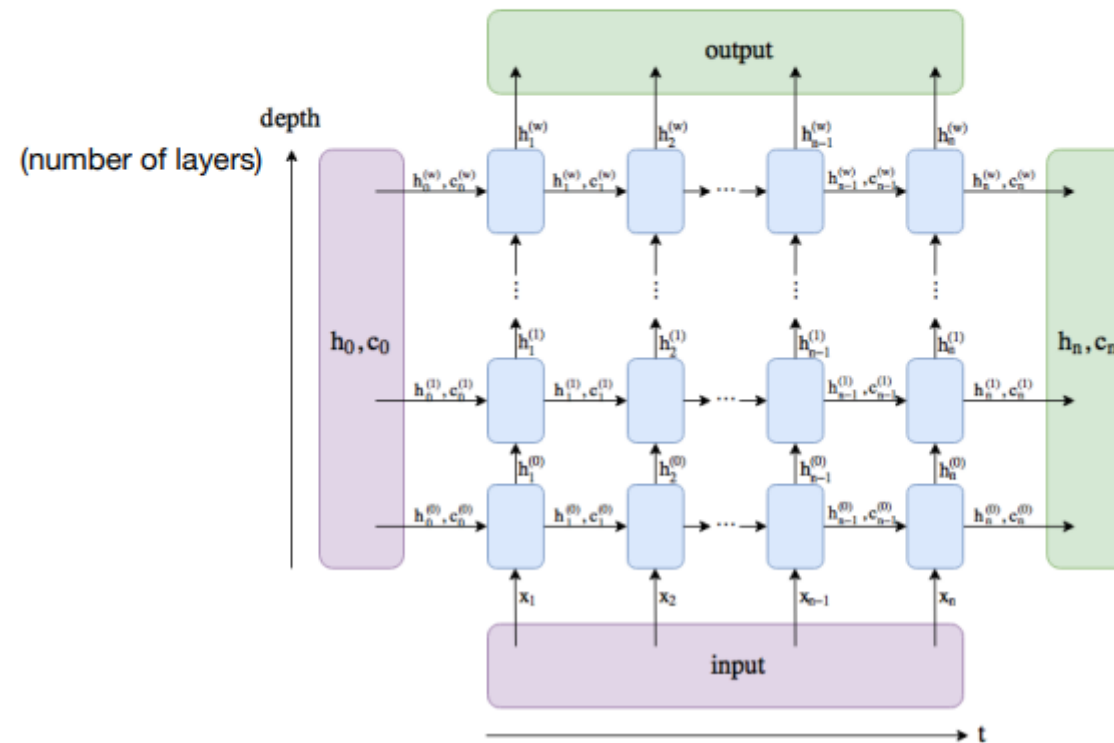
Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

Clase LSTM

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```



Clase LSTM

<https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html>

Inputs: input, (h_0, c_0)

- **input** of shape $(batch, input_size)$: tensor containing input features
 - **h_0** of shape $(batch, hidden_size)$: tensor containing the initial hidden state for each element in the batch.
 - **c_0** of shape $(batch, hidden_size)$: tensor containing the initial cell state for each element in the batch.
- If (h_0, c_0) is not provided, both **h_0** and **c_0** default to zero.

Outputs: (h_1, c_1)

- **h_1** of shape $(batch, hidden_size)$: tensor containing the next hidden state for each element in the batch
- **c_1** of shape $(batch, hidden_size)$: tensor containing the next cell state for each element in the batch

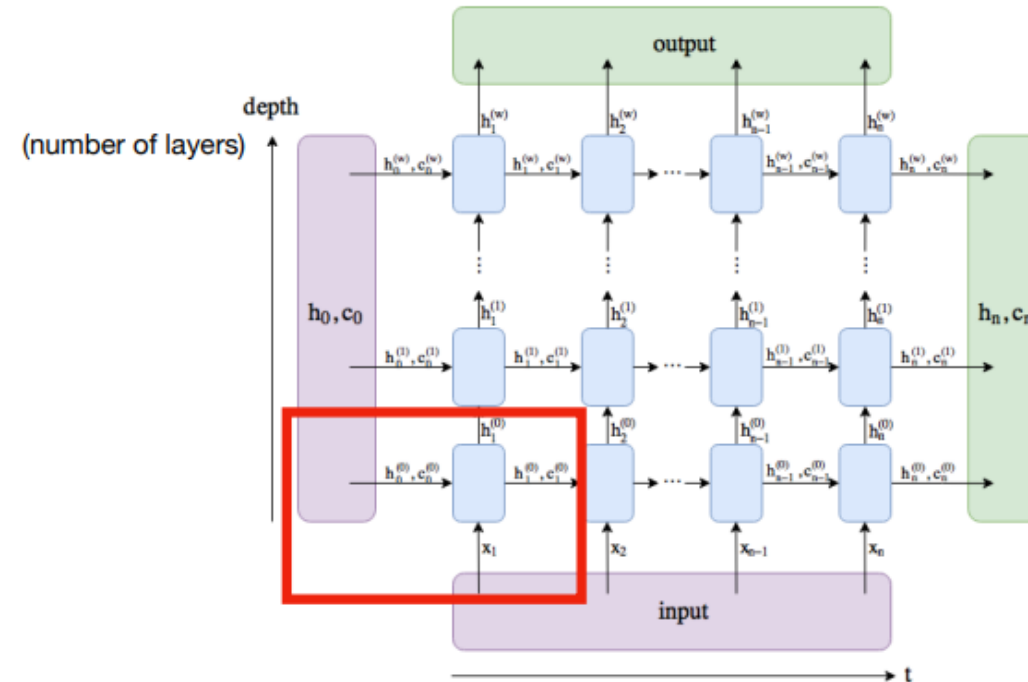
Examples:

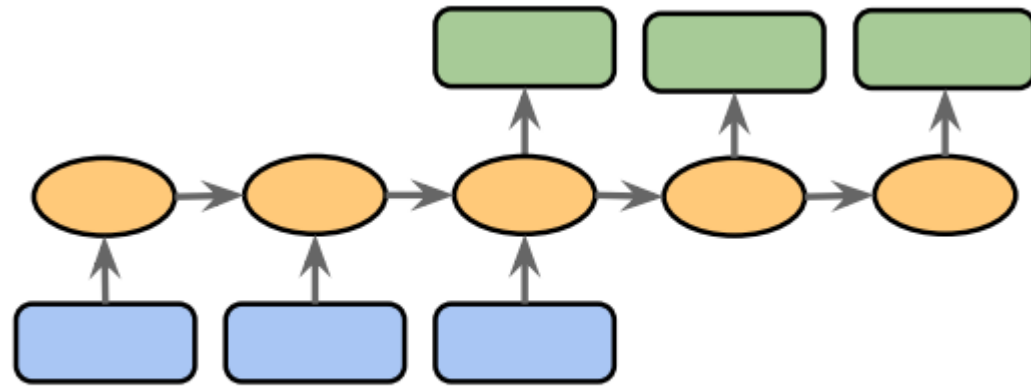
```
>>> rnn = nn.LSTMCell(10, 20) # (input_size, hidden_size)
>>> input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
>>> hx = torch.randn(3, 20) # (batch, hidden_size)
>>> cx = torch.randn(3, 20)
>>> output = []
>>> for i in range(input.size()[0]):
>>>     hx, cx = rnn(input[i], (hx, cx))
>>>     output.append(hx)
>>> output = torch.stack(output, dim=0)
```

Clase LSTM

Examples:

```
>>> rnn = nn.LSTMCell(10, 20) # (input_size, hidden_size)
>>> input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
>>> hx = torch.randn(3, 20) # (batch, hidden_size)
>>> cx = torch.randn(3, 20)
>>> output = []
>>> for i in range(input.size()[0]):
>>>     hx, cx = rnn(input[i], (hx, cx))
>>>     output.append(hx)
>>> output = torch.stack(output, dim=0)
```





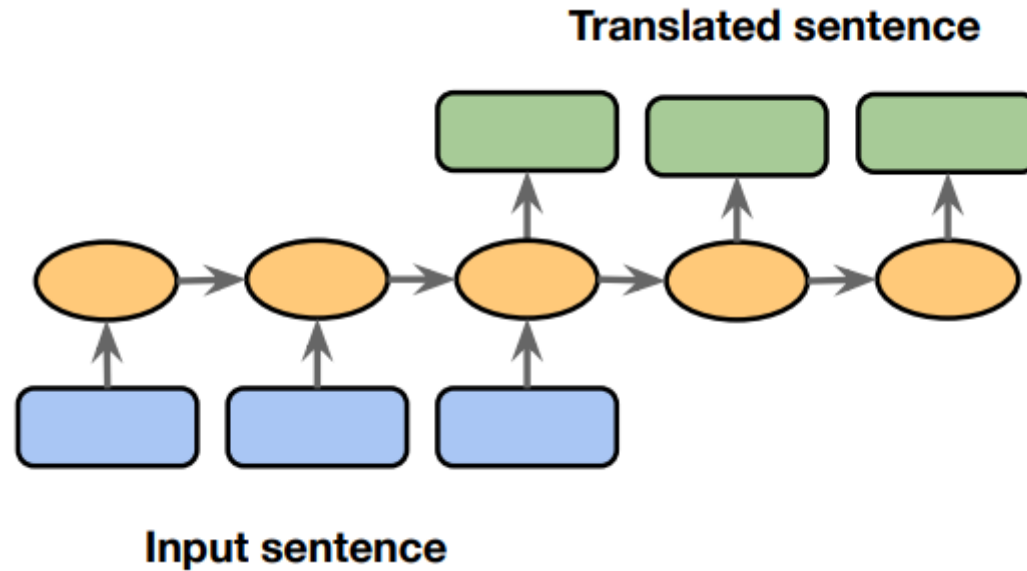
many-to-many

Traducción con una secuencia a secuencia red y atención (inglés a francés)
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

RNN de varios a varios para generar texto

1. - Generación de secuencias con RNN
2. - Caracter RNN en PyTorch
3. - **RNN con atención**
4. - Atención es todo lo que necesitamos
5. - Modelos de transformadores
6. - Transformador en PyTorch

Arquitectura de varios a varios para la traducción de idiomas



Traducción con una secuencia a secuencia red y atención (inglés a francés)
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Input:

Today is a great day



Translation:

Heute ist ein großartiger Tag

Input:

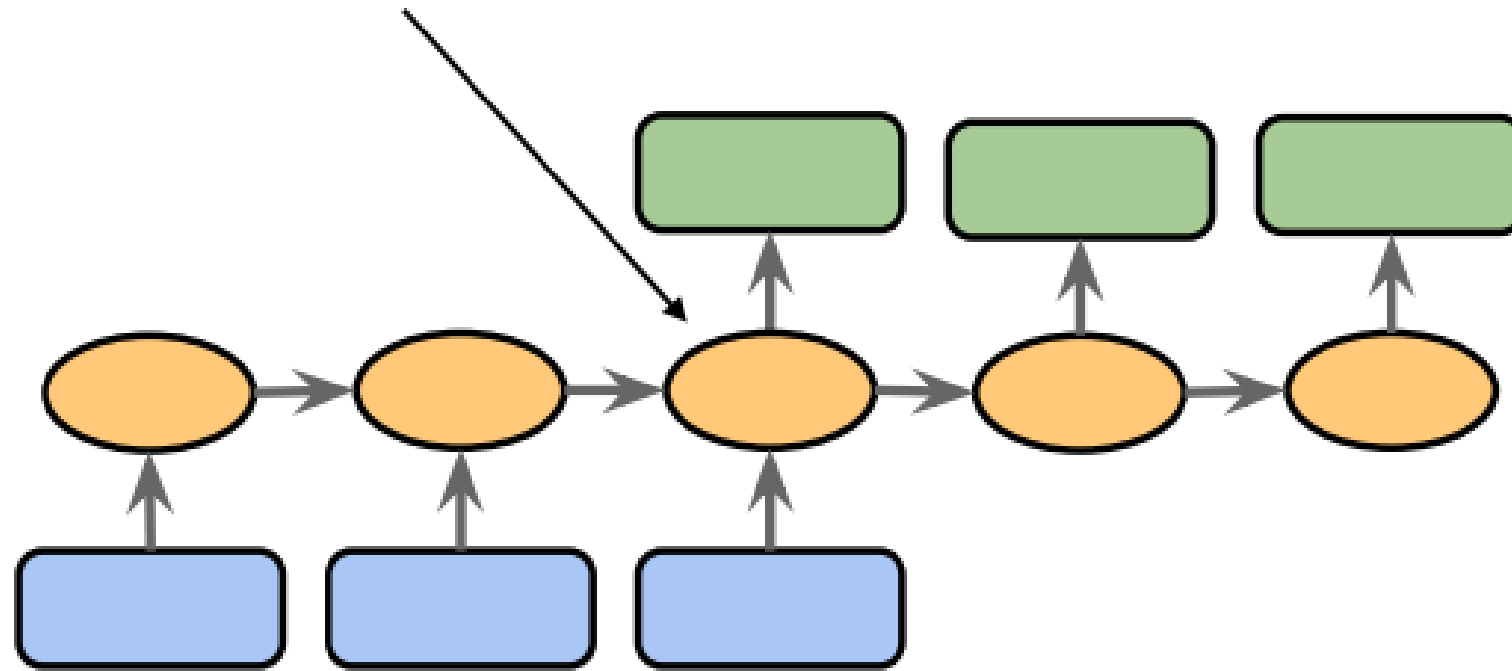
If you've ever studied a foreign language, you've probably encountered a "false friend" at some point.



Translation:

Wenn Sie jemals eine Fremdsprache gelernt haben, sind Sie wahrscheinlich irgendwann auf einen „falschen Freund“ gestoßen.

Desafío en la traducción de
idiomas: memorice la oración
de entrada completa en un
estado oculto



many-to-many

Mecanismo de atención

- Desarrollado originalmente para la traducción de idiomas: Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. <https://arxiv.org/abs/1409.0473>

"...permitir que un modelo busque automáticamente (de forma suave) partes de una oración de origen que sean relevantes para predecir una palabra de destino..."

"traditional"
encoder+decoder
RNN

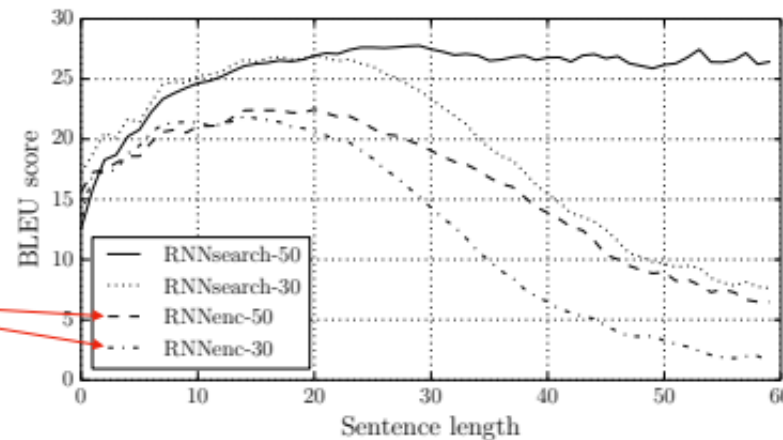


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

Mecanismo de atención

Asigne un peso de atención a cada palabra, para saber cuánta "atención" debe prestar el modelo a cada palabra (es decir, para cada palabra, la red aprende un "contexto")

Mecanismo de atención

- Desarrollado originalmente para la traducción de idiomas: Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.
<https://arxiv.org/abs/1409.0473>

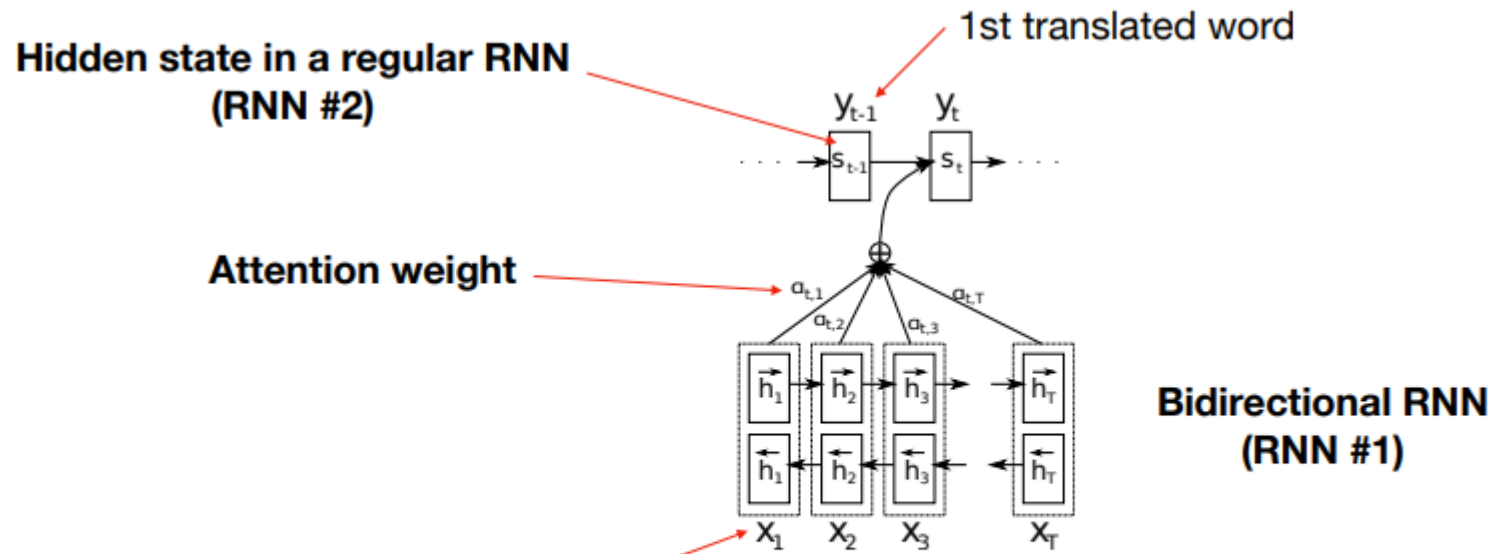


Figure 1: The graphical illustration of the proposed model trying to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) .

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

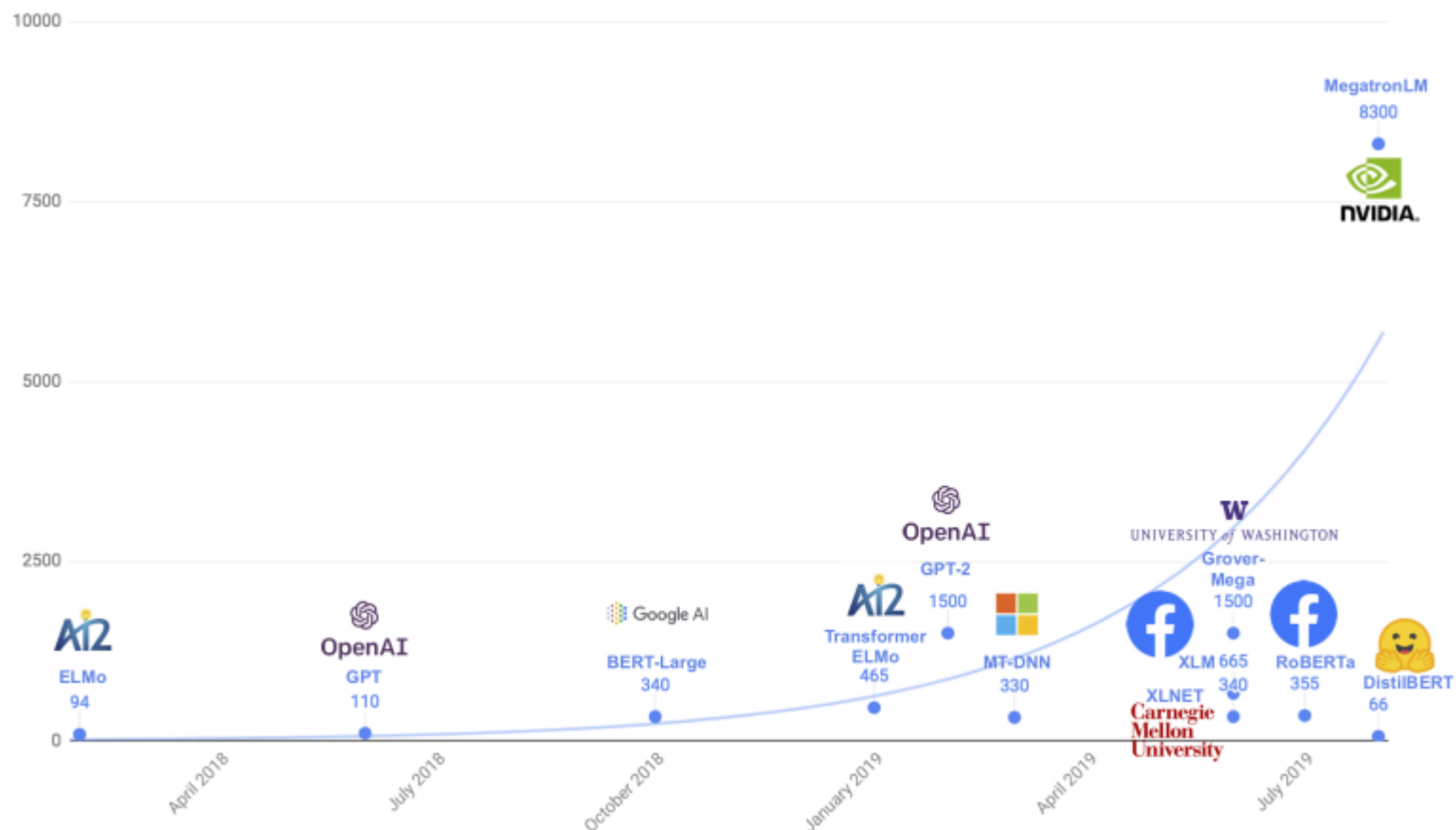
Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

<https://arxiv.org/abs/1706.03762>

Desde ~ 2018, Los transformadores han ido creciendo en popularidad... y tamaño



Mecanismo de autoatención - forma muy básica

Procedimiento principal:

- 1) Derivar pesos de atención: Similitud entre la entrada actual y todas las demás entradas (siguiente diapositiva)
- 2) Normalizar pesos a través de softmax (siguiente diapositiva)
- 3) Calcule el valor de atención a partir de pesos normalizados y las entradas correspondientes (a continuación)

Autoatención como suma ponderada:

$$A_i = \sum_{j=1}^T a_{ij} x_j$$

salida correspondiente a la i-ésima entrada

peso basado en la similitud entre la entrada actual x_i y todas las demás entradas

Mecanismo de autoatención - forma muy básica

Autoatención como suma ponderada:

$$\text{salida correspondiente a la } i\text{-ésima entrada} \rightarrow \mathbf{A}_i = \sum_{j=1}^T a_{ij} \mathbf{x}_j$$

peso basado en la similitud
entre la entrada actual \mathbf{x}_i
y todas las demás entradas

¿Cómo calcular los pesos
de atención?:

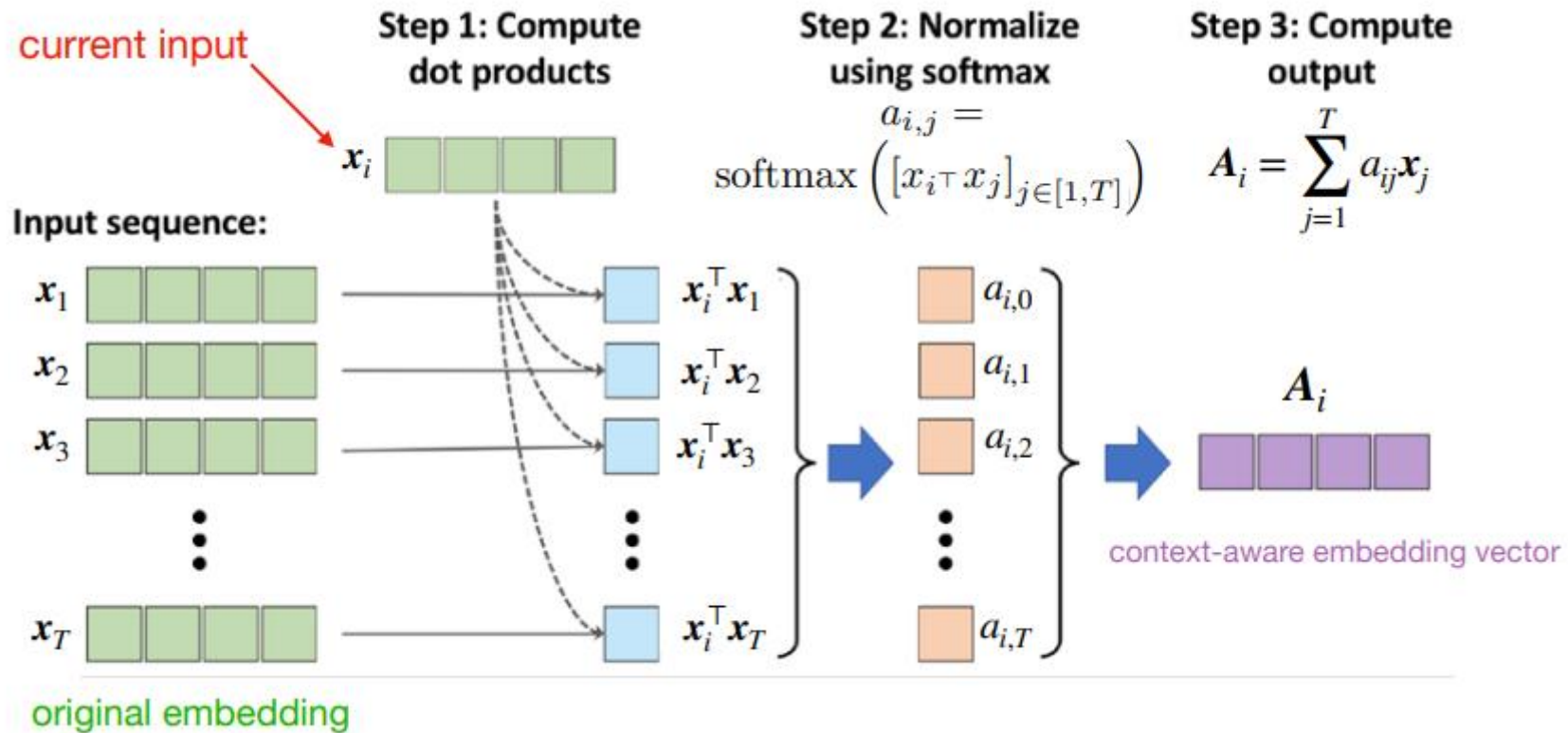
Aquí como producto punto simple:

$$e_{ij} = \mathbf{x}_i^T \mathbf{x}_j$$

Repita esto para todas las entradas $j \in \{1 \dots T\}$, luego normalizar

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{j=1}^T \exp(e_{ij})} = \text{softmax}([e_{ij}]_{j=1 \dots T})$$

Mecanismo de autoatención - forma muy básica

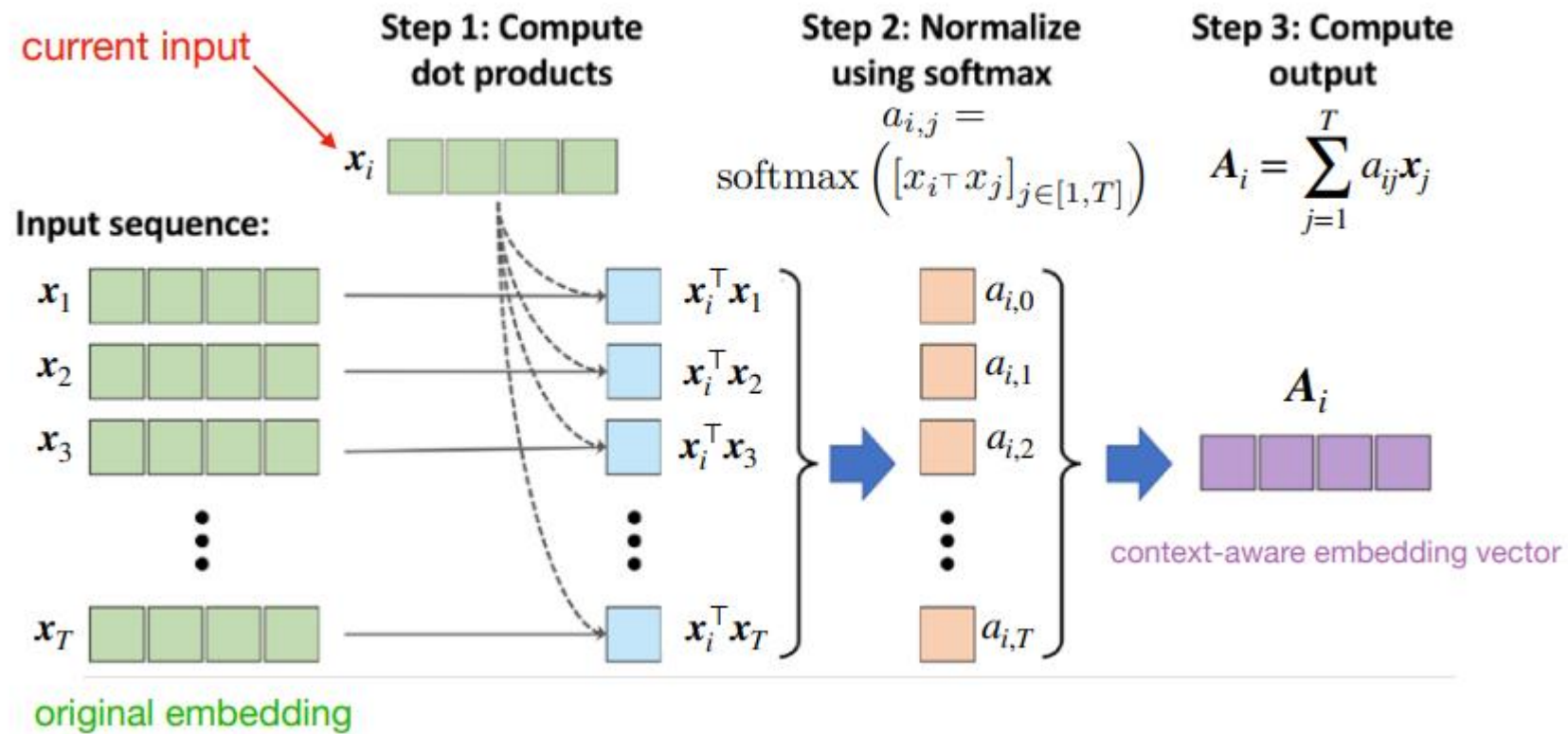


Usar la atención sin el RNN: transformadores y mecanismo de autoatención

1. Generación de secuencias con RNN
2. Caracter RNN en PyTorch
3. RNN con atención
- 4. Atención es todo lo que necesitamos**
 - 4.1 Forma básica de autoatención
 - 4.2 Atención personal y atención de productos punto**
 - 4.3 Atención multicabezal
5. Modelos de transformadores
 - 5.1 La arquitectura del transformador
 - 5.2 Algunos modelos de transformadores populares: BERT, GPT y BART
6. Transformador en PyTorch

Mecanismo de autoatención - forma muy básica

Autoatención: Relacionar diferentes posiciones dentro de una sola secuencia (vs. entre secuencias de entrada y salida)



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
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

<https://arxiv.org/abs/1706.03762>

Mecanismo de autoatención

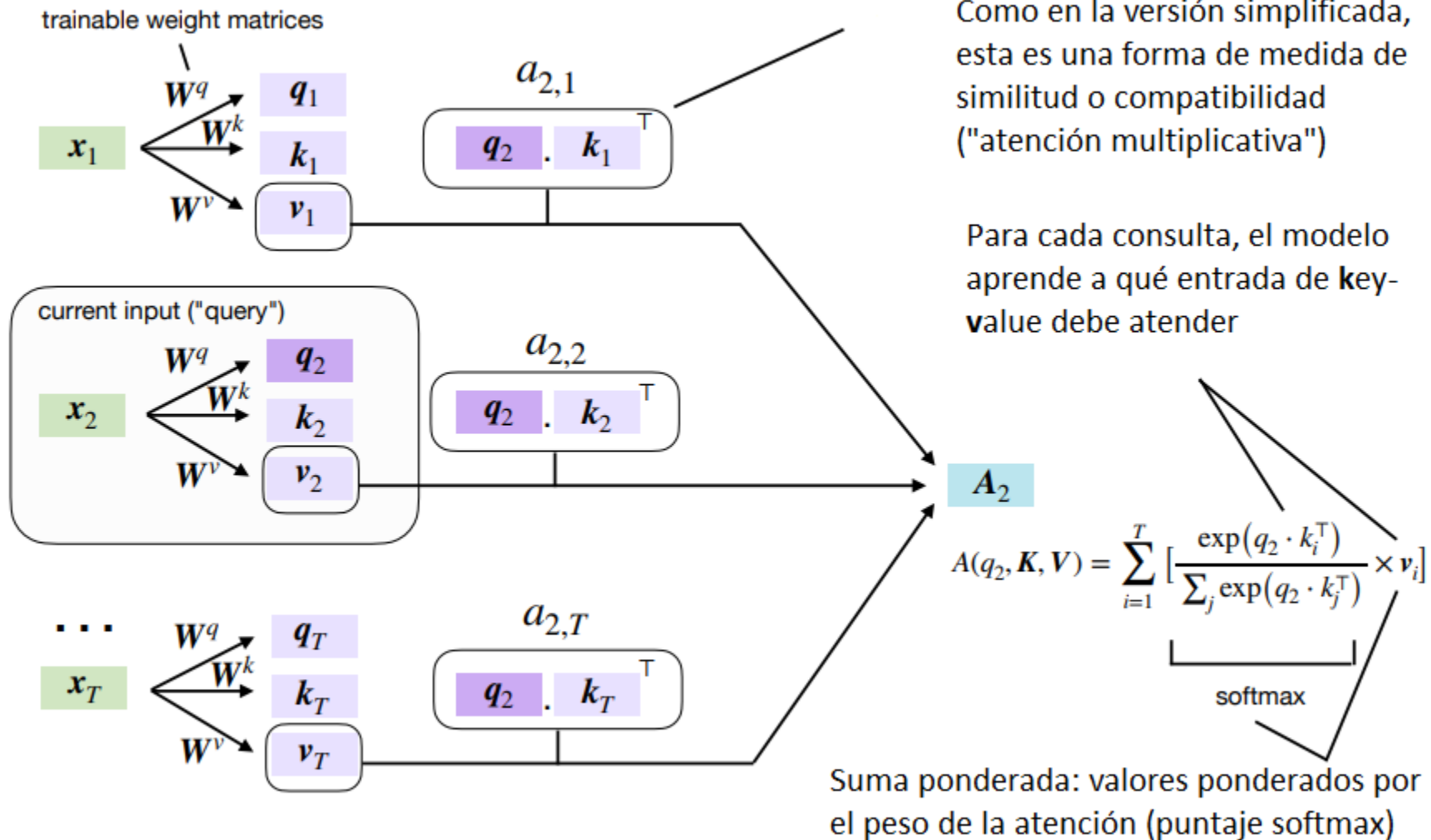
- La versión básica anterior no incluía ningún parámetro que se pudiera aprender, por lo que no es muy útil para aprender un modelo de lenguaje.
- Ahora estamos agregando 3 matrices de peso entrenables que se multiplican con las incrustaciones de la secuencia de entrada $(x_i's)$

$$\text{query} = W^q x_i$$

$$\text{key} = W^k x_i$$

$$\text{value} = W^v x_i$$

Mecanismo de autoatención

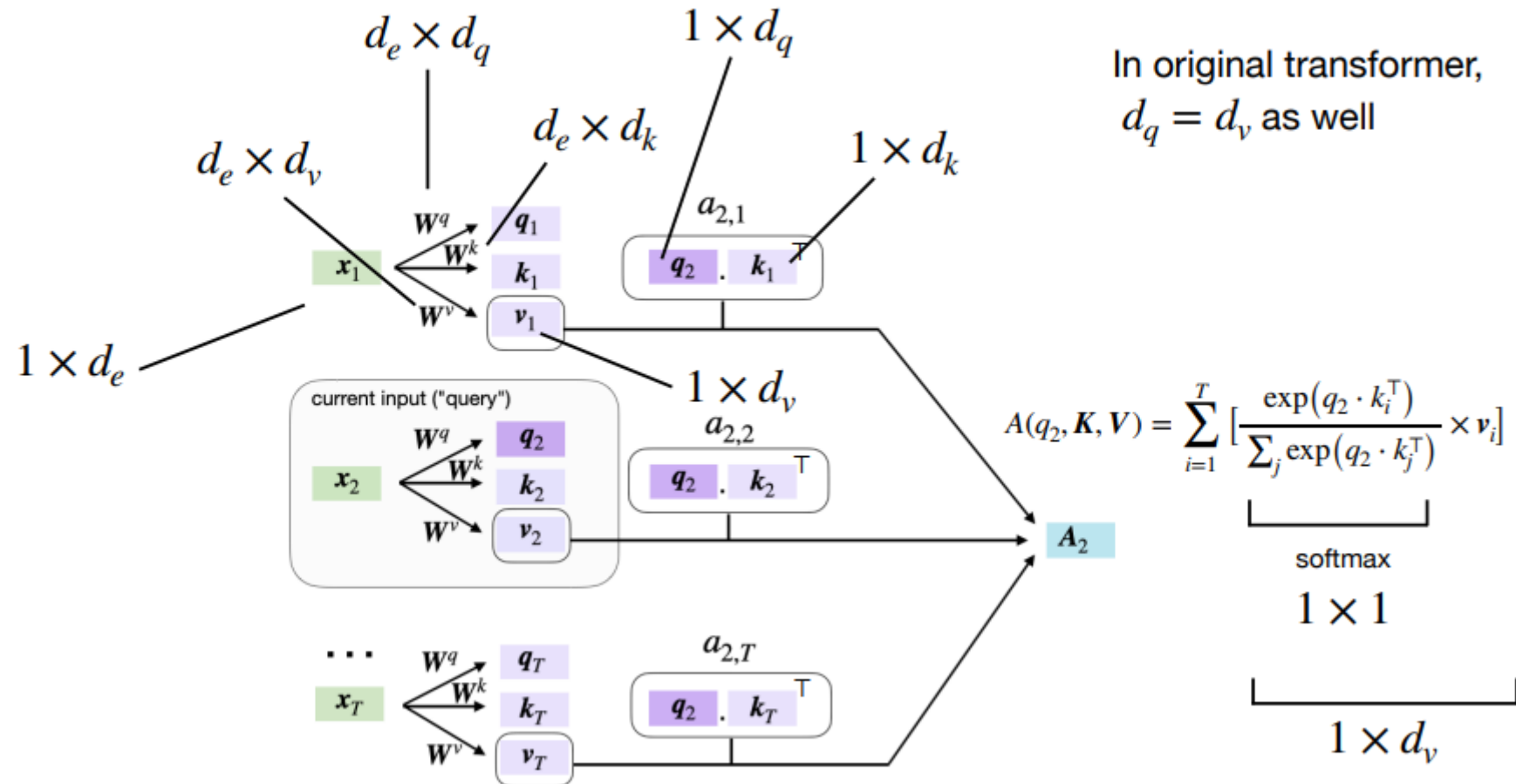


Mecanismo de autoatención

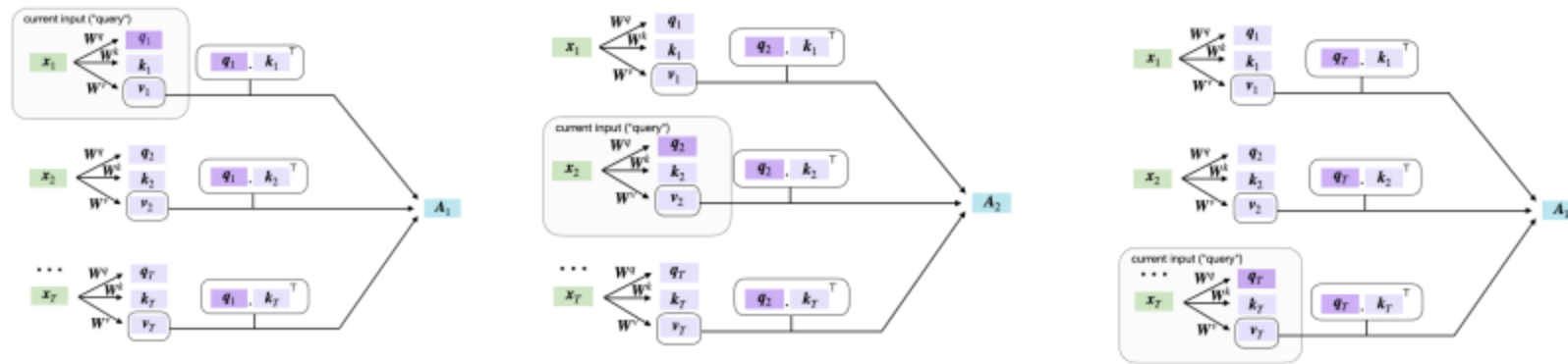
d_e = embedding size (original transformer = 512)

where $d_q = d_k$

In original transformer,
 $d_q = d_v$ as well



Mecanismo de autoatención



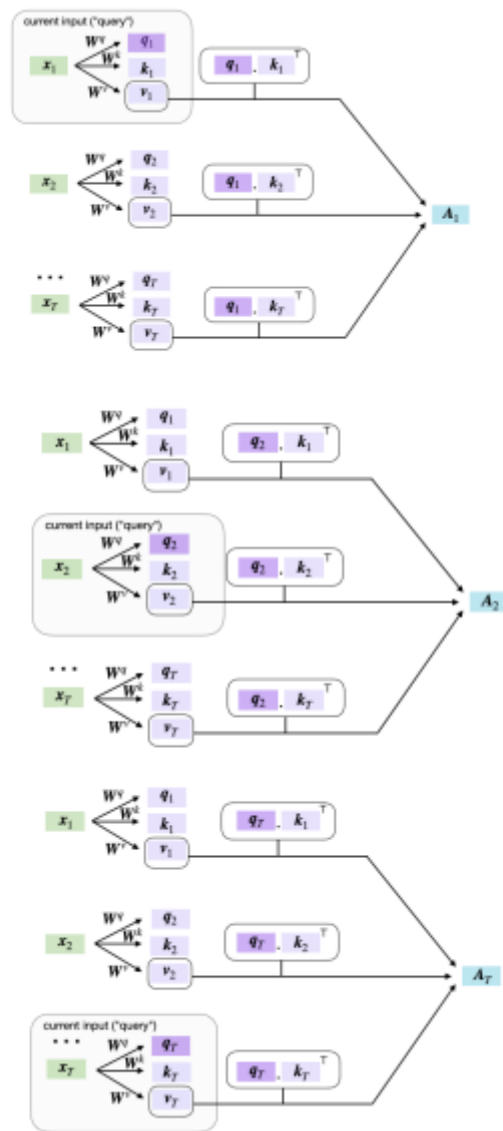
Matriz de puntuación de atención: $\mathbf{A} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}$

Mecanismo de autoatención (atención de productos punto escalados)

d_e = embedding size

T = input sequence size

$$\mathbf{x} \in \mathbb{R}^{T \times d_e}$$



$$\mathbf{Q} \in \mathbb{R}^{T \times d_q}$$

$$\mathbf{K} \in \mathbb{R}^{T \times d_k}$$

$$\mathbf{V} \in \mathbb{R}^{T \times d_v}$$

"attention matrix"

$$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \underbrace{\text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \right)}_{T \times T} \mathbf{V}$$

$T \times d_v$

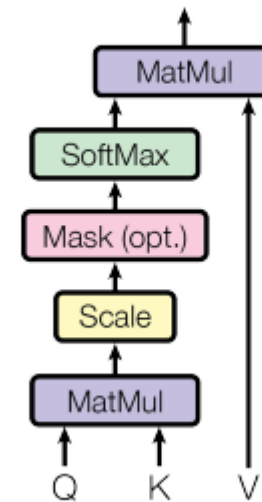
"attention-based embedding"

Mecanismo de autoatención

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Para asegurarse de que los productos punto entre la consulta y la clave y no crezcan demasiado (y el gradiente de softmax se vuelva demasiado pequeño) para d_k grandes

Scaled Dot-Product Attention



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need

Usar la atención sin el RNN: transformadores y mecanismo de autoatención

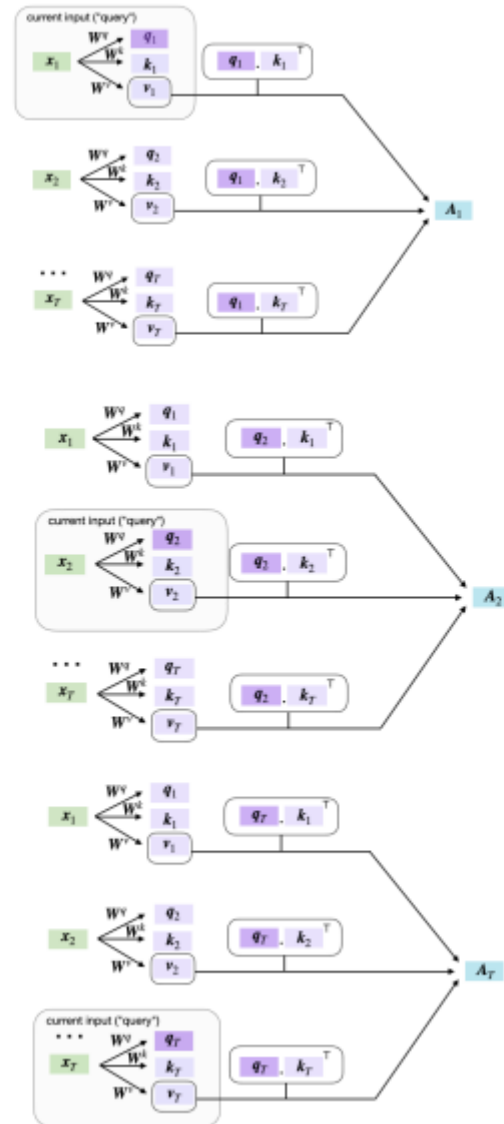
1. Generación de secuencias con RNN
2. Caracter RNN en PyTorch
3. RNN con atención
- 4. Atención es todo lo que necesitamos**
 - 4.1 Forma básica de autoatención
 - 4.2 Atención personal y atención de productos punto
 - 4.3 Atención multicabezal**
5. Modelos de transformadores
 - 5.1 La arquitectura del transformador
 - 5.2 Algunos modelos de transformadores populares: BERT, GPT y BART
6. Transformador en PyTorch

Mecanismo de autoatención (atención de productos punto escalados)

d_e = embedding size

T = input sequence size

$$x \in \mathbb{R}^{T \times d_e}$$



$$Q \in \mathbb{R}^{T \times d_q}$$

$$K \in \mathbb{R}^{T \times d_k}$$

$$V \in \mathbb{R}^{T \times d_v}$$

"attention matrix"

$$T \times T$$

$$A(Q, K, V) = \left[\text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) \right]$$

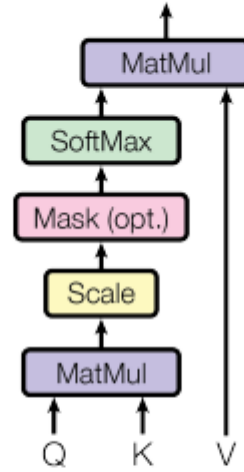
$$T \times d_v$$

"attention-based embedding"

Atención multicable

- Aplicar la autoatención varias veces en paralelo (similar a varios núcleos para canales en CNN)
- Para cada cabeza (capa de autoatención), usar diferentes W^q, W^k, W^v , y concatenar los resultados, A_i
- 8 cabezas de atención en el transformador original, es decir,
 $W_{(1)}^q, W_{(1)}^k, W_{(1)}^v, \dots, W_{(8)}^q, W_{(8)}^k, W_{(8)}^v$
- Permite atender diferentes partes de la secuencia de manera diferente

Scaled Dot-Product Attention



Multi-Head Attention

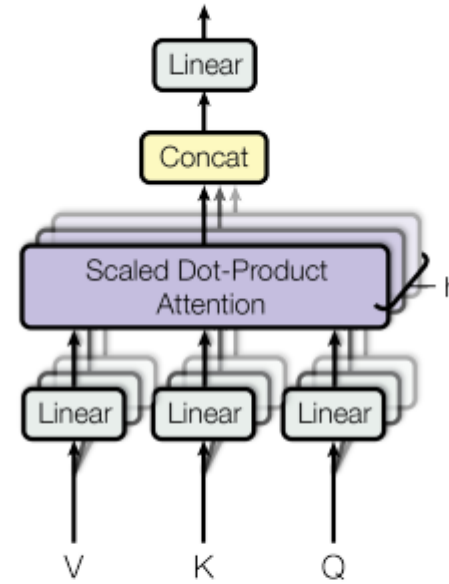
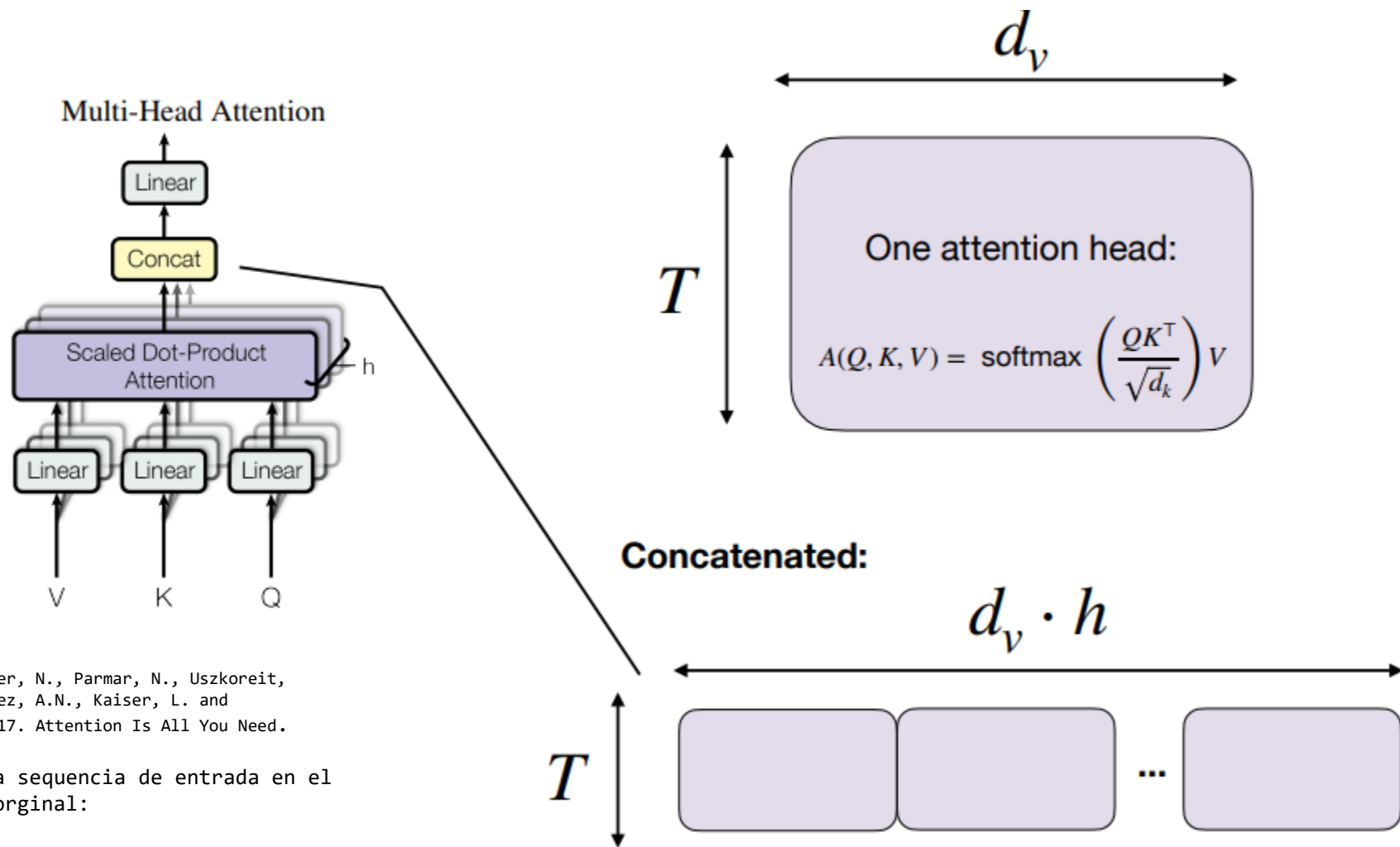


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.



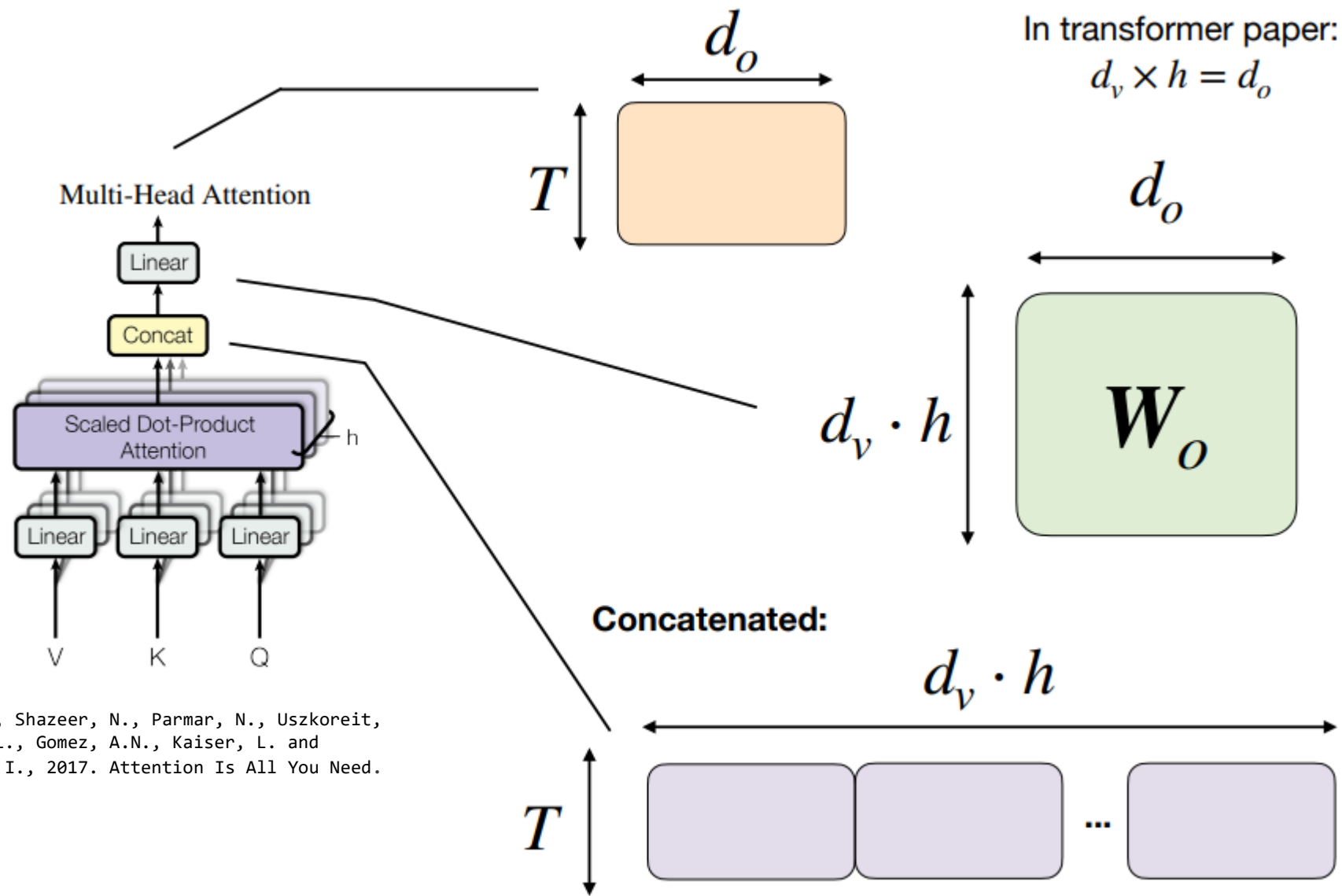
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Dimensión de la secuencia de entrada en el transformador original:

$$T \times d_e = T \times 512$$

y

$$d_v = 512/h = 64$$



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Usar la atención sin el RNN: transformadores y mecanismo de autoatención

1. Generación de secuencias con RNN
2. Caracter RNN en PyTorch
3. RNN con atención
4. Atención es todo lo que necesitamos
 - 4.1 Forma básica de autoatención
 - 4.2 Atención personal y atención de productos punto
 - 4.3 Atención multicabezal
5. **Modelos de transformadores**
 - 5.1 **La arquitectura del transformador**
 - 5.2 Algunos modelos de transformadores populares: BERT, GPT y BART
6. Transformador en PyTorch

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

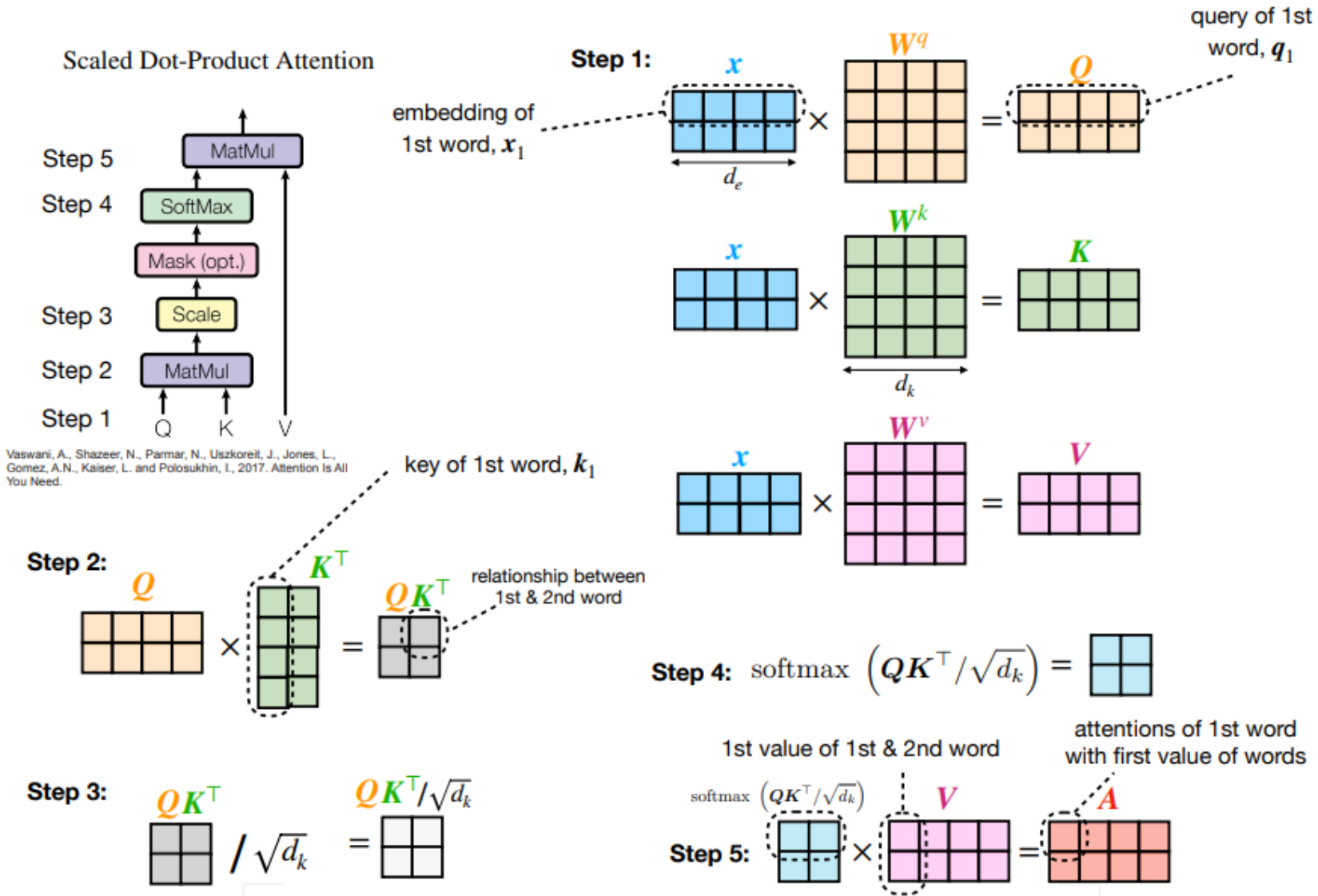
Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

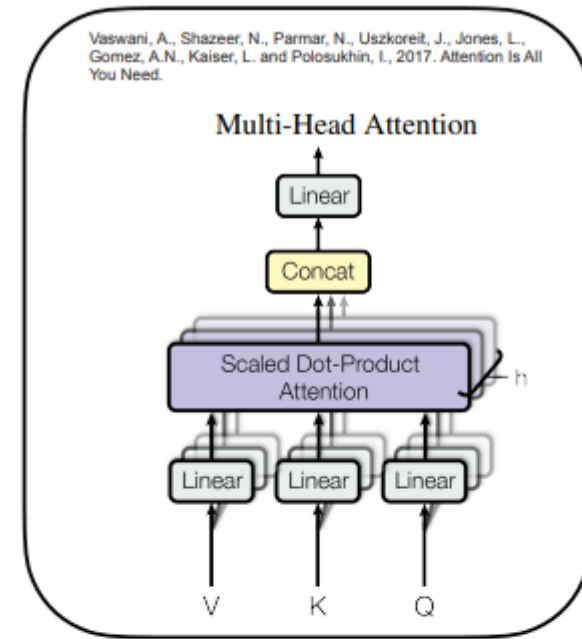
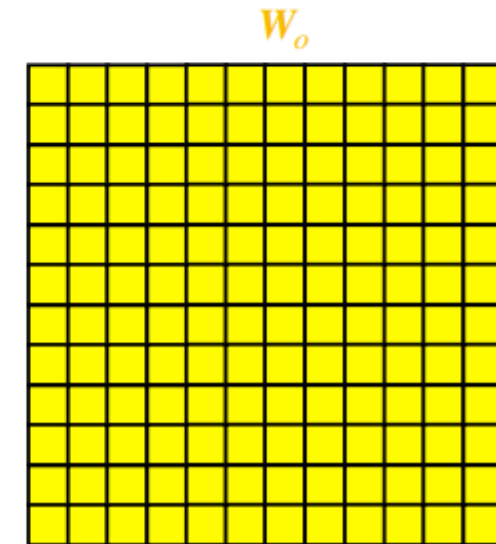
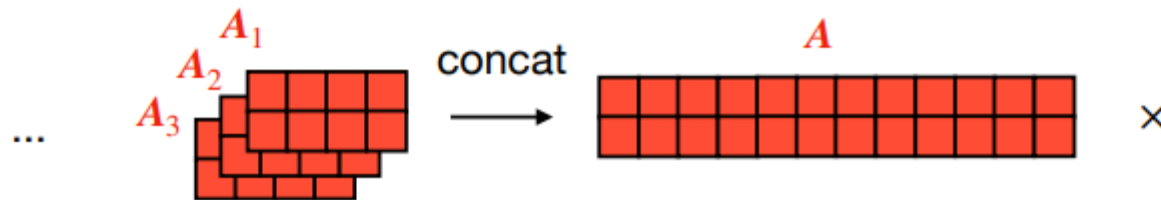
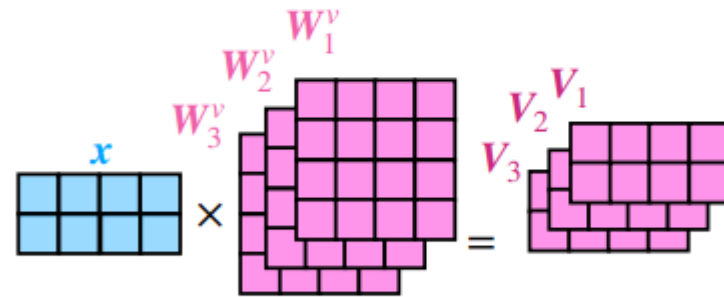
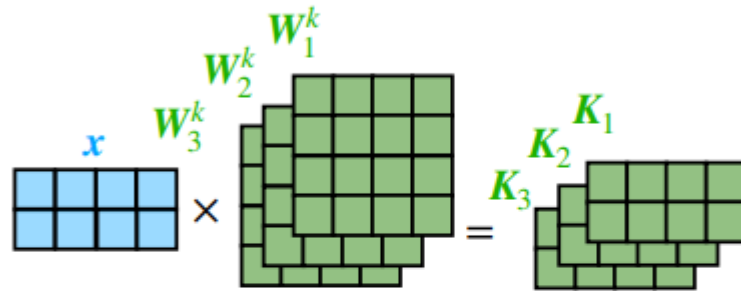
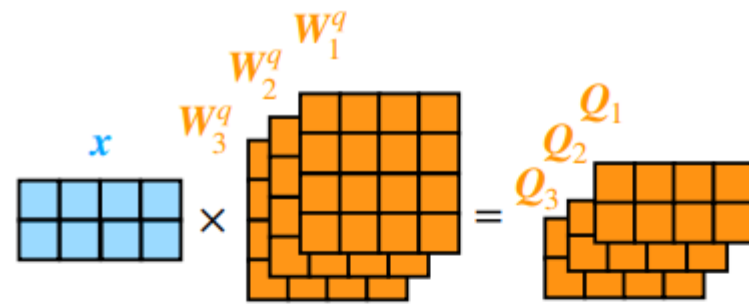
Illia Polosukhin* ‡
illia.polosukhin@gmail.com

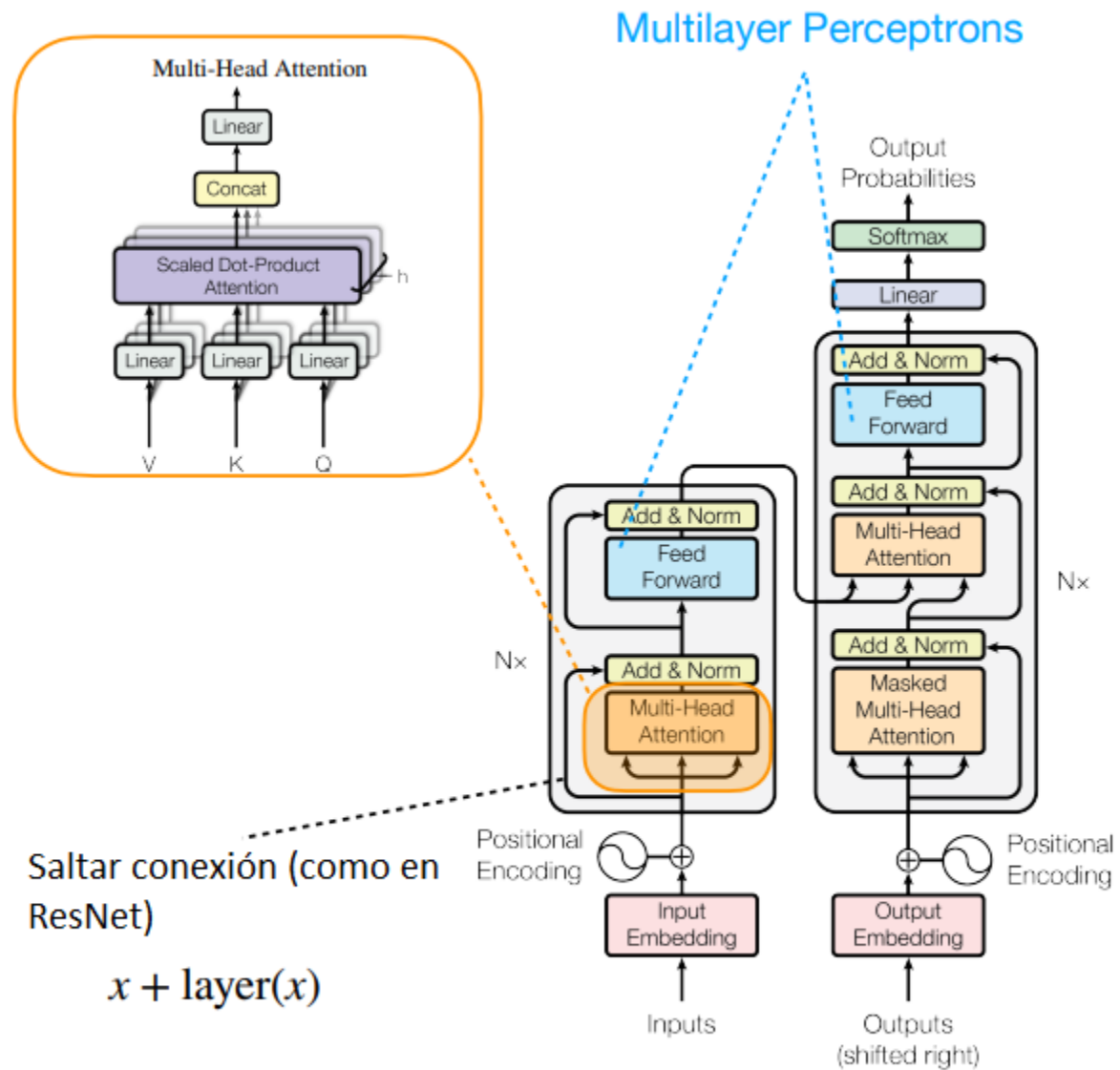
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

<https://arxiv.org/abs/1706.03762>

Resumen de atención de productos punto escalados







Genera palabras de salida una a la vez

Figure 1: The Transformer - model architecture.

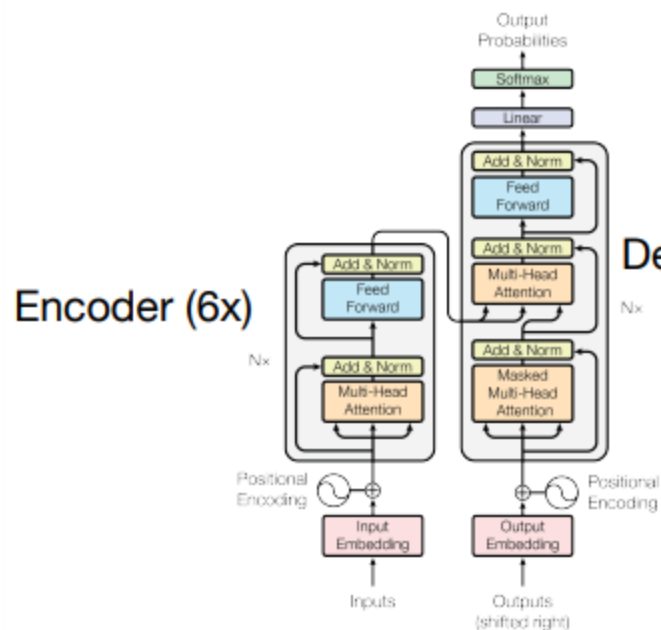
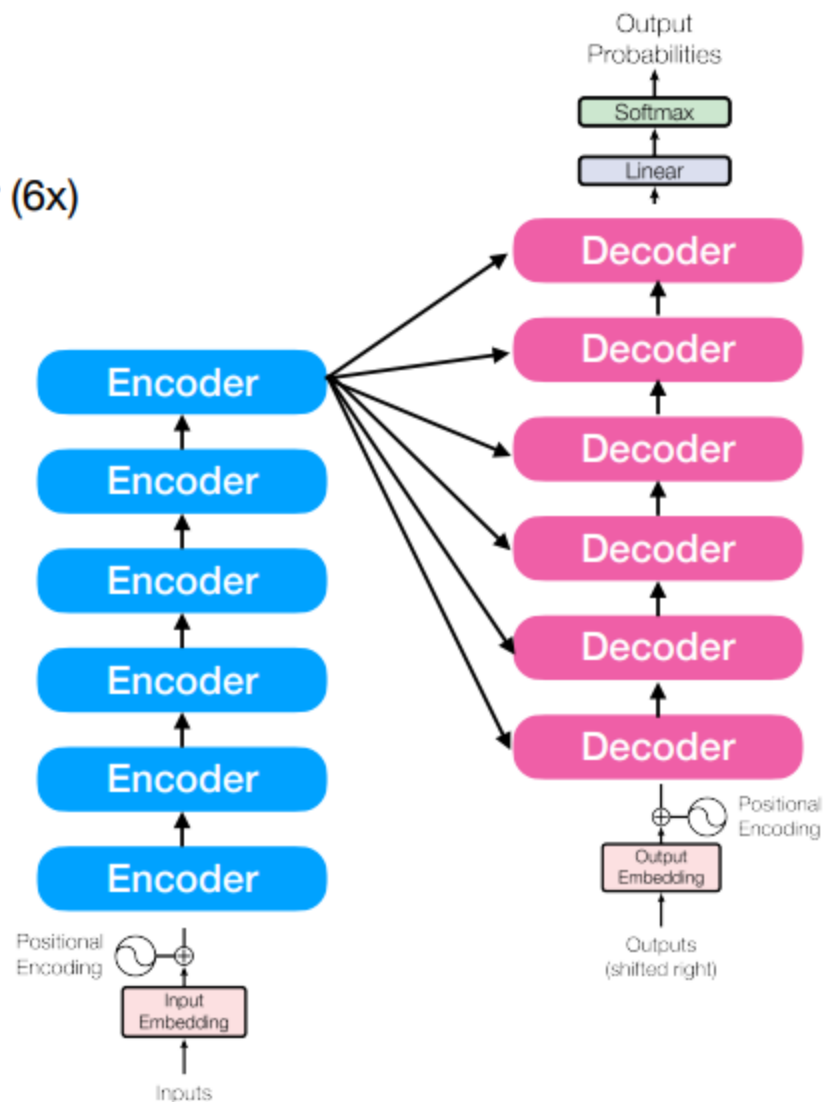
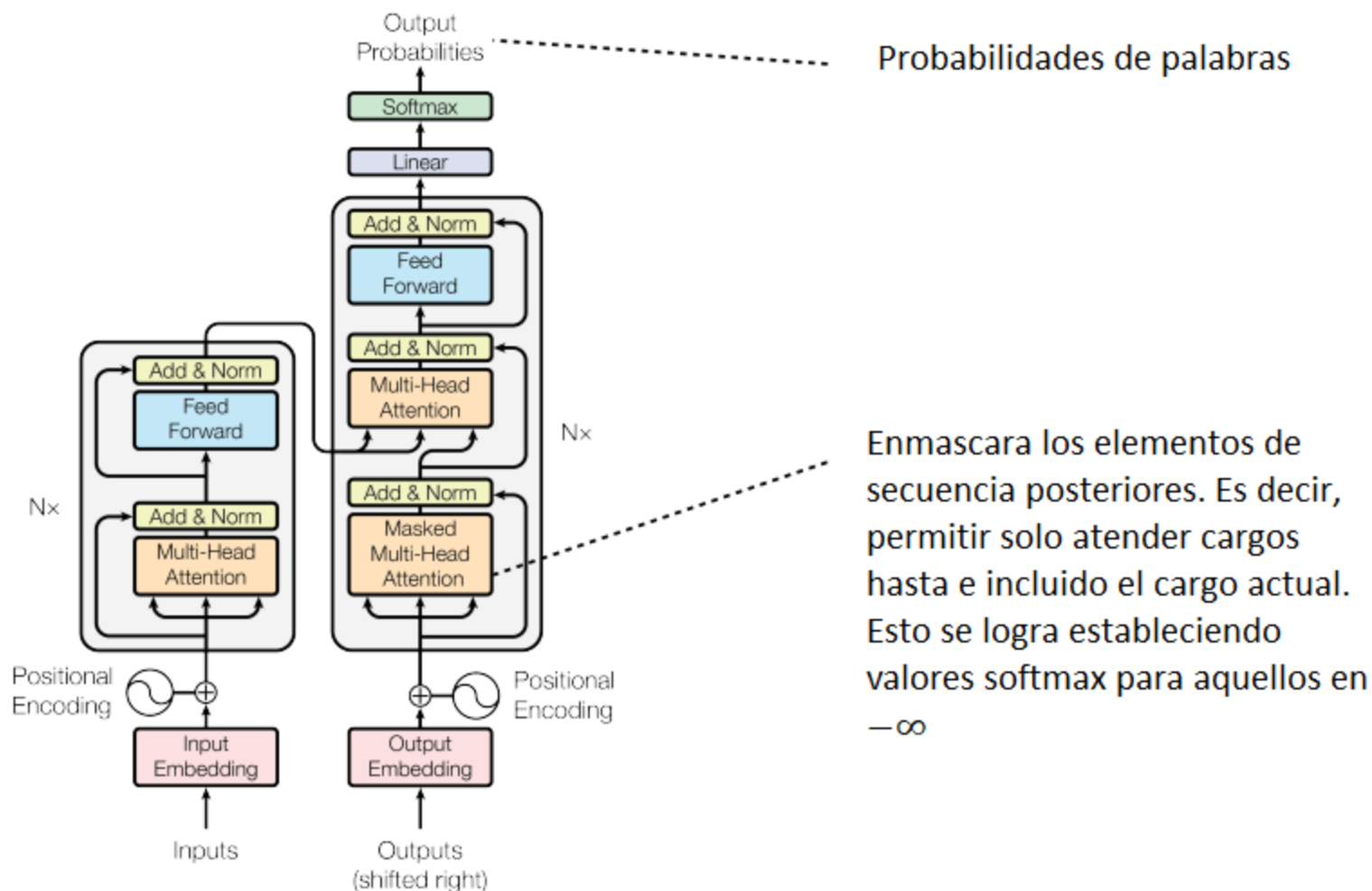


Figure 1: The Transformer - model architecture.

Misma estructura y
dimensión de entrada/salida
para cada codificador y
decodificador



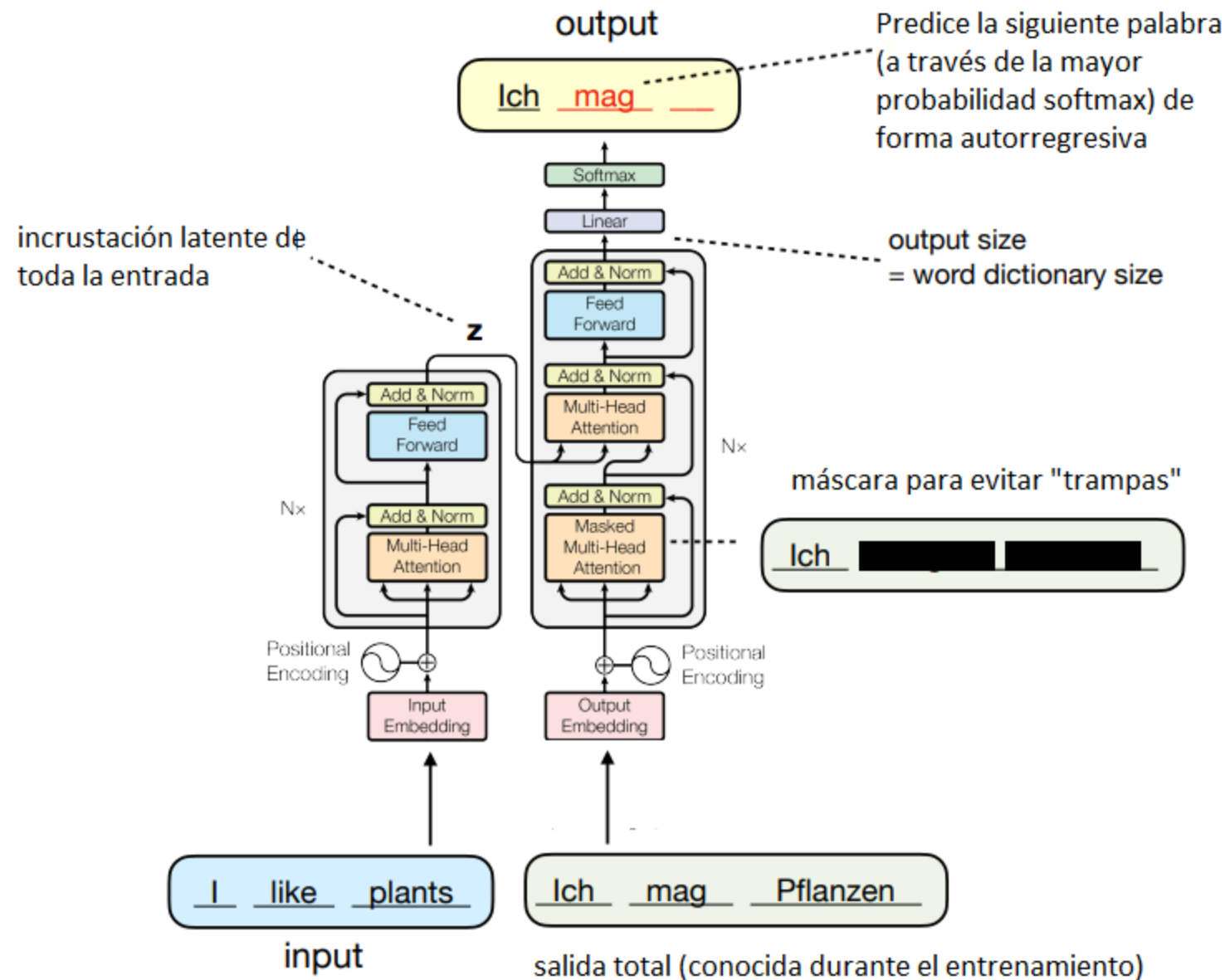
Atención multicabezal enmascarada



Enmascara los elementos de secuencia posteriores. Es decir, permitir solo atender cargos hasta e incluido el cargo actual. Esto se logra estableciendo valores softmax para aquellos en $-\infty$

Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.



Agregar matriz de codificación posicional a la matriz de incrustación de palabras

- El producto punto escalado y la capa completamente conectada no varían en la permutación
- La codificación posicional sinusoidal es un vector de pequeños valores (constantes) agregados a las incrustaciones
- Como resultado, la misma palabra tendrá incrustaciones ligeramente diferentes dependiendo de dónde se encuentren en la oración.

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{(2i+1)/d_{\text{model}}}}\right)$$

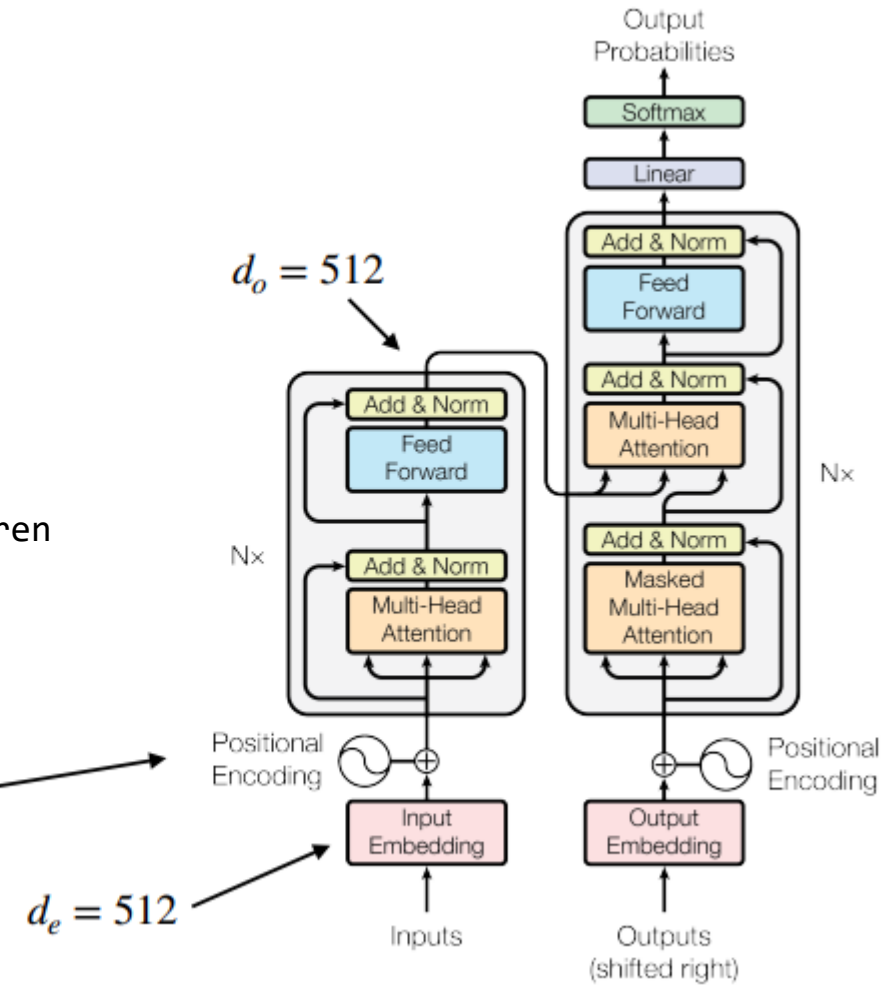


Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

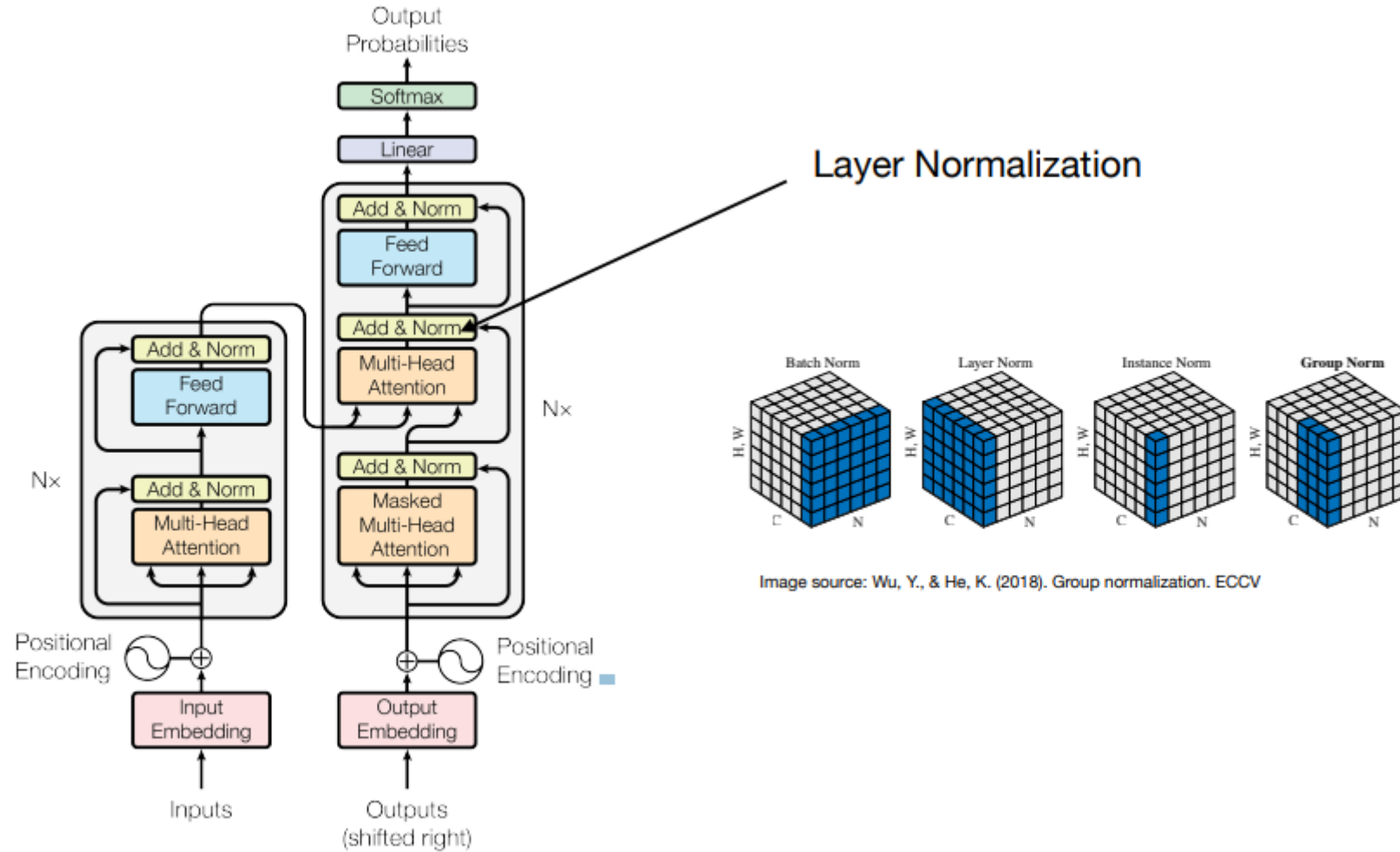


Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

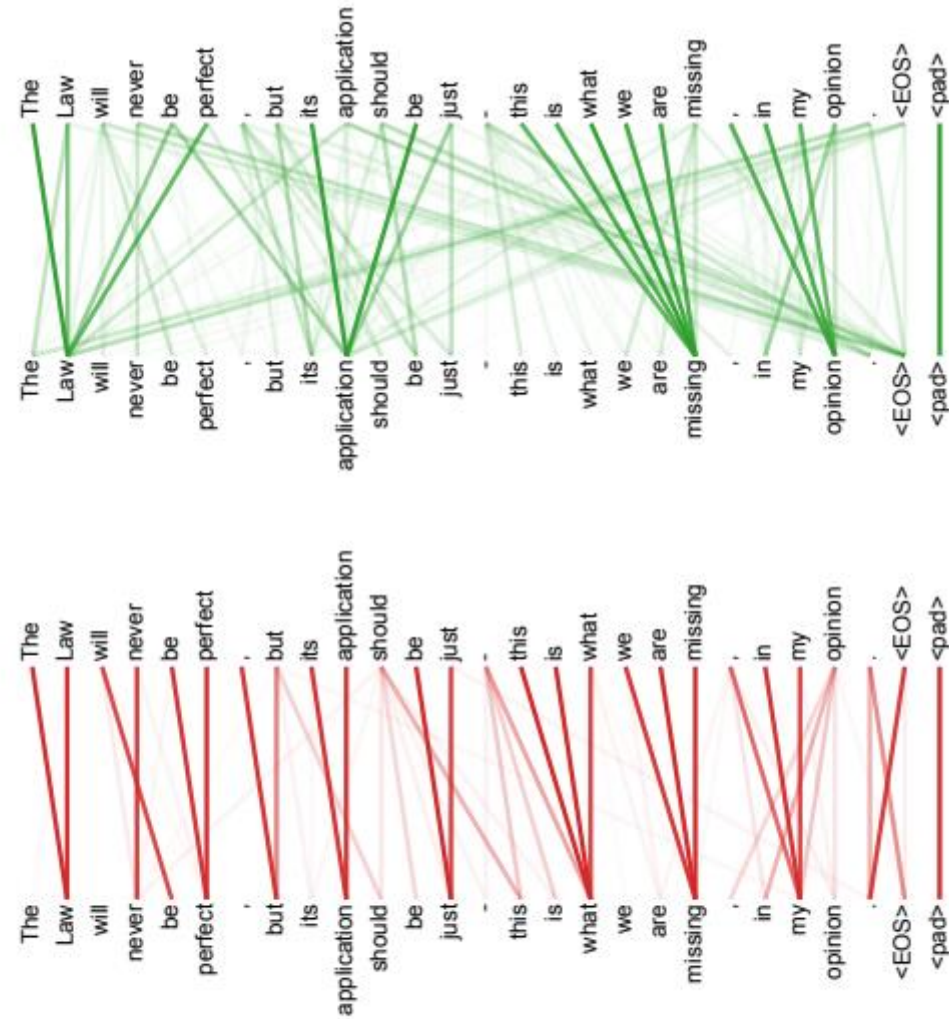


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Usar la atención sin el RNN: transformadores y mecanismo de autoatención

1. Generación de secuencias con RNN
2. Caracter RNN en PyTorch
3. RNN con atención
4. Atención es todo lo que necesitamos
 - 4.1 Forma básica de autoatención
 - 4.2 Atención personal y atención de productos punto
 - 4.3 Atención multicabezal
5. **Modelos de transformadores**
 - 5.1 La arquitectura del transformador
 - 5.2 **Algunos modelos de transformadores populares: BERT, GPT y BART**
6. Transformador en PyTorch

Resumen

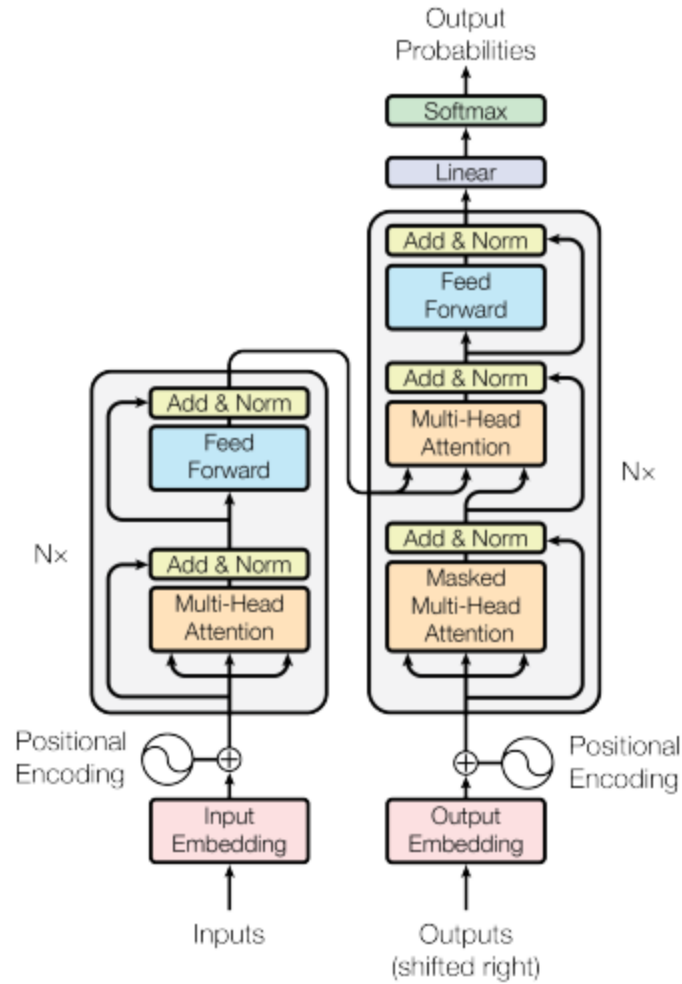


Figure 1: The Transformer - model architecture.

Las dos claves del éxito detrás de transformadores

1. Autoatención para codificar dependencias de largo alcance
2. Autosupervisión para aprovechar grandes conjuntos de datos sin etiquetar

Enfoque de entrenamiento de transformadores

1. Entrenamiento previo en grandes conjuntos de datos sin etiquetar (aprendizaje auto-supervisado)
2. Capacitación para tareas posteriores sobre datos etiquetados (aprendizaje supervisado)
 - a) Enfoque de ajuste fino
 - b) Enfoque basado en características

Algunos modelos de transformadores populares: BERT, GPT y BART

- 5.2.2 GPT-v1: Transformador generativo pre-entrenado
- 5.2.3 BERT: Representaciones de codificador bidireccional de transformadores
- 5.2.4 GPT-v2: Los modelos de lenguaje son aprendices multitarea sin supervisión
- 5.2.5 GPT-v3: Los modelos de lenguaje son aprendices con pocas posibilidades
- 5.2.6 BART: Combinando transformadores bidireccionales y autorregresivos
- 5.2.7 Palabras de cierre: el reciente crecimiento de los transformadores del lenguaje

Algunos modelos de transformadores populares: BERT, GPT y BART

GPT-v1:
Transformador generativo pre-entrenado

GPT-v1: Transformador generativo pre-entrenado

- Desarrollado por OpenAI
- Unidireccional: entrenado para predecir la siguiente palabra en una oración

GPT (110 million parameters)

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2 (1.5 billion parameters)

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.
https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3 (175 billion parameters)

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. <https://arxiv.org/abs/2005.14165>

Conceptos clave de GPT-v1

- Cuello de botella: falta de datos etiquetados
- 2 - Proceso de entrenamiento en 2 pasos ("semi-supervisado")
 1. Entrenamiento previo generativo (sobre datos sin etiquetar); aprendizaje no supervisado/"autosupervisado"
 2. Ajuste fino discriminativo (en datos etiquetados), aprendizaje supervisado
- Entrenamiento previo sobre un gran conjunto de datos BookCorpus (7000 libros)
- Basado en la arquitectura del decodificador del Transformador original ("Atención es todo lo que necesita")

Arquitectura de GPT-v1 y tareas posteriores

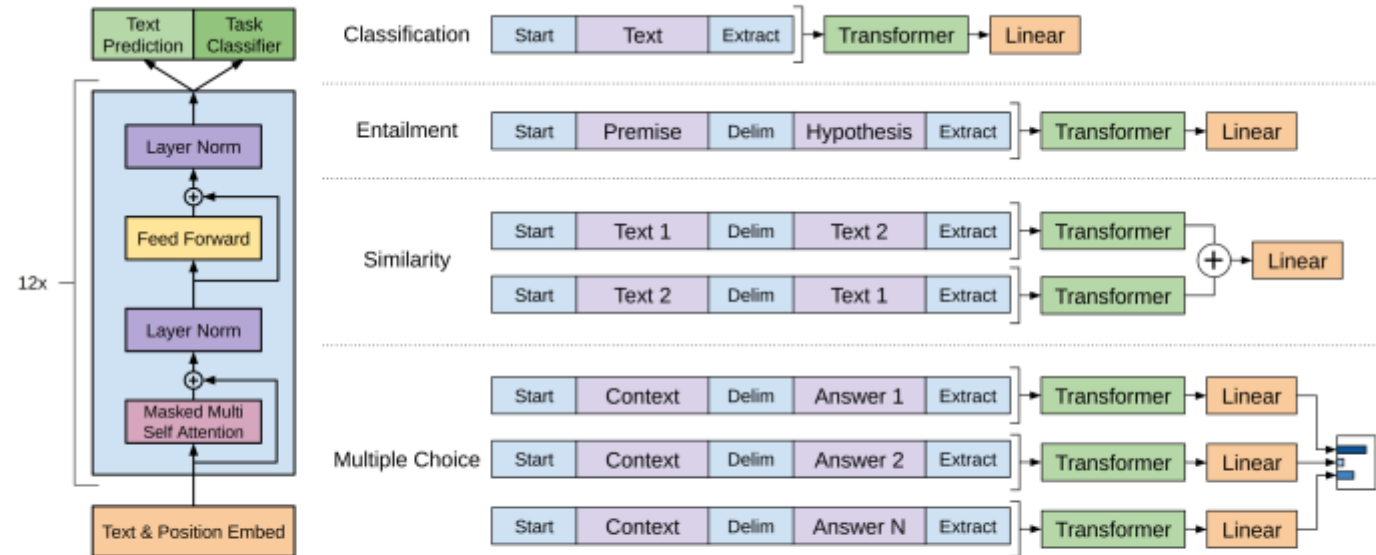


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Estudio de ablación GPT-v1

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	70.3	81.8	88.1	56.0
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	75.0	47.9	92.0	84.9	83.2	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

Algunos modelos de transformadores populares:
BERT, GPT y BART

**BERT: Representaciones de codificador bidireccional
de transformadores**

BERT (Bidirectional Encoder Representations from Transformers)

Paper: Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding.

<https://arxiv.org/abs/1810.04805>

(Google Research, 2018)

- Codificador de transformador bidireccional multicapa
- Arquitectura casi idéntica al transformador original y GPT, excepto
 - Enmascaramiento bidireccional (conocido como tarea "Cloze", Taylor 1953 *)
 - Predicción de la siguiente oración como tarea adicional de preentrenamiento

Entradas de BERT

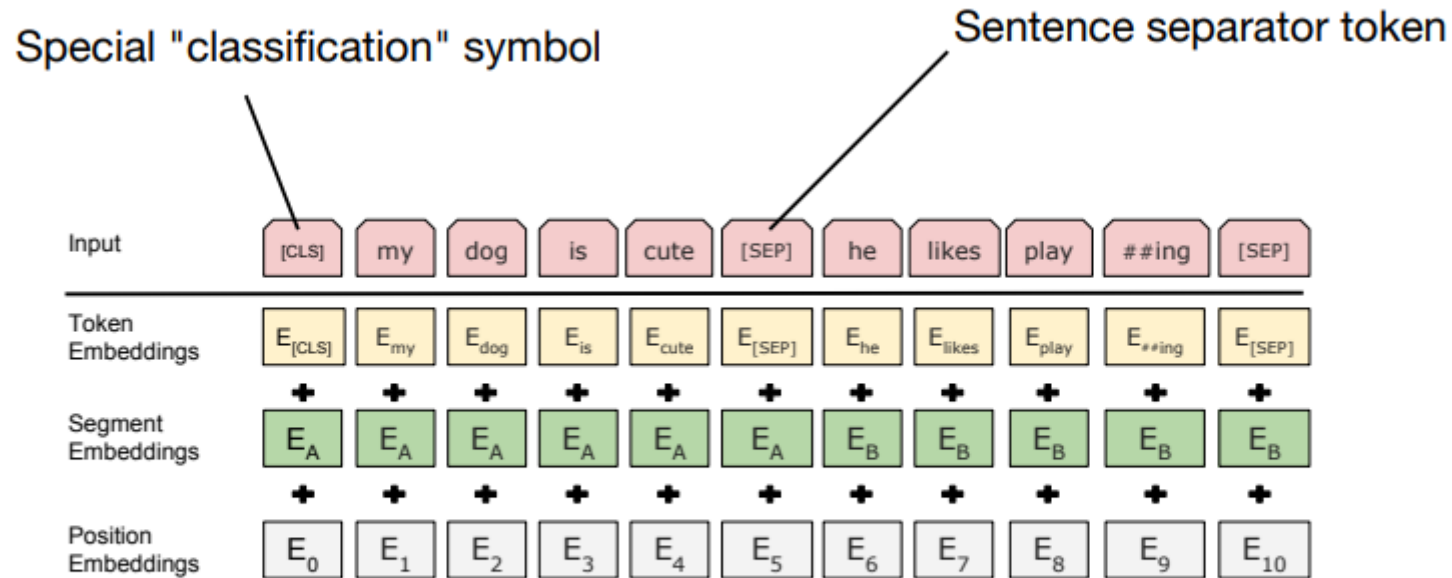


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Las incrustaciones de tokens son incrustaciones de WordPiece* con un tamaño de vocabulario de 30.000

* Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, Klingner J. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144. 2016
<https://arxiv.org/abs/1609.08144>

Tareas previas al entrenamiento de BERT

Conjuntos de datos previos al entrenamiento

- BookCorpus (800 millones de palabras)
- Wikipedia (2500 millones de palabras)

Tareas previas al entrenamiento

- Modelo de lenguaje enmascarado ("Cloze")
- Predicción de la siguiente oración

Tarea 1 de preentrenamiento BERT - Modelo de lenguaje enmascarado

Sentencia de entrada: A quick brown *fox* jumps over the lazy dog

"Marca" el 15% de las palabras:

- 80%: Reemplaza con [*Máscara*]
- 10%: Reemplaza con palabra aleatoria (*café*)
- 10%: Dejar como está (*fox*) para imitar el escenario de ajuste fino

Tarea 1 de preentrenamiento BERT - Modelo de lenguaje enmascarado

Sentencia de entrada: A quick brown *fox* jumps over the lazy dog



Enmascarado al azar: A quick brown [*Máscara*] jumps over the lazy dog



BERT



Possible classes
(all words)

0.2%	ant
...	...
11%	fox
...	...
0.01%	zoo

Tarea 2 de preentrenamiento BERT - Predicción de la siguiente oración

Tarea de clasificación binaria equilibrada (50% IsNext, 50% NotNext)

Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

Tareas previas y posteriores al entrenamiento de BERT

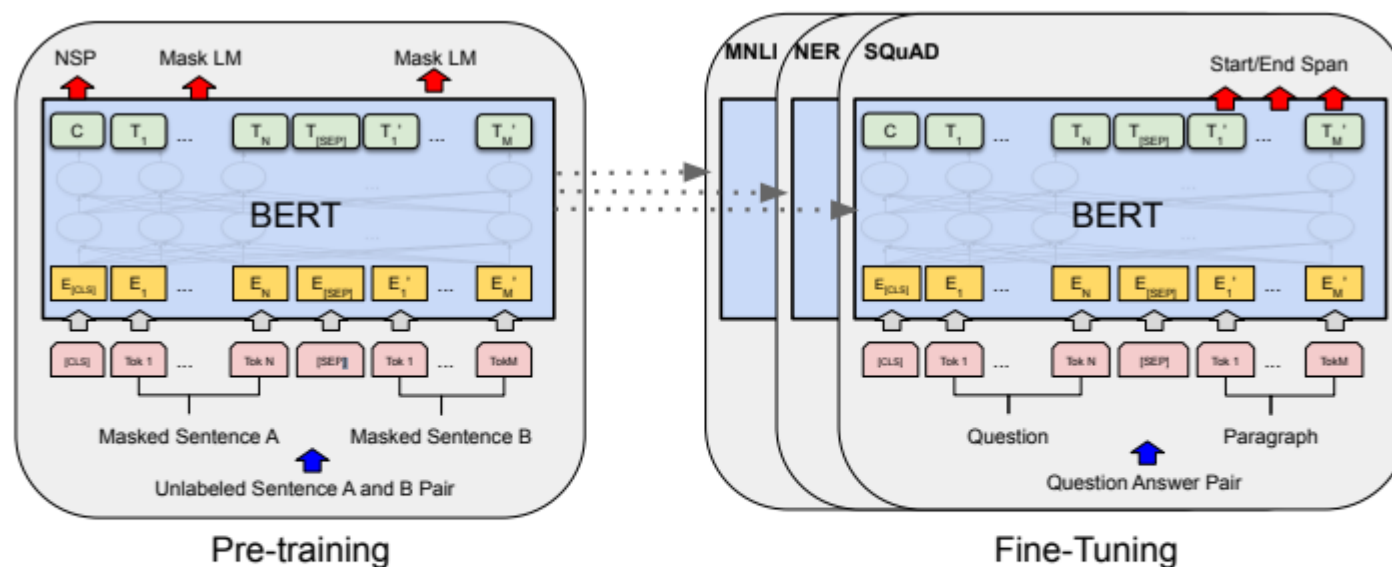


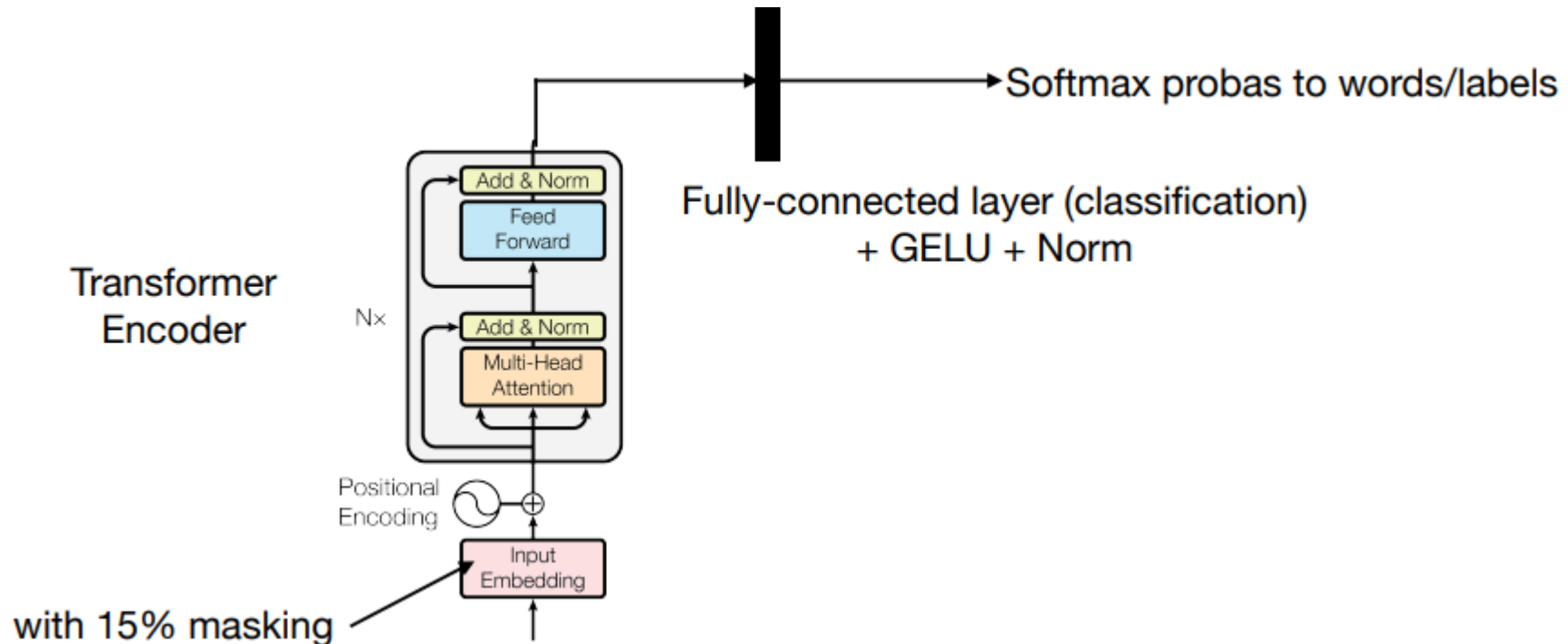
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

Enfoque de entrenamiento de transformadores

1. **Entrenamiento previo** en grandes conjuntos de datos sin etiquetar (aprendizaje auto-supervisado)
2. **Capacitación para tareas posteriores** sobre datos etiquetados (aprendizaje supervisado)
 - a) Enfoque de ajuste fino
 - b) Enfoque basado en características (hoy en día también llamado "ajuste fino")

Enfoque de preentrenamiento y ajuste de BERT

- Agregar capa de clasificación
- Entrene de un extremo a otro en un conjunto de datos etiquetado para la tarea posterior (actualice TODOS los parámetros)



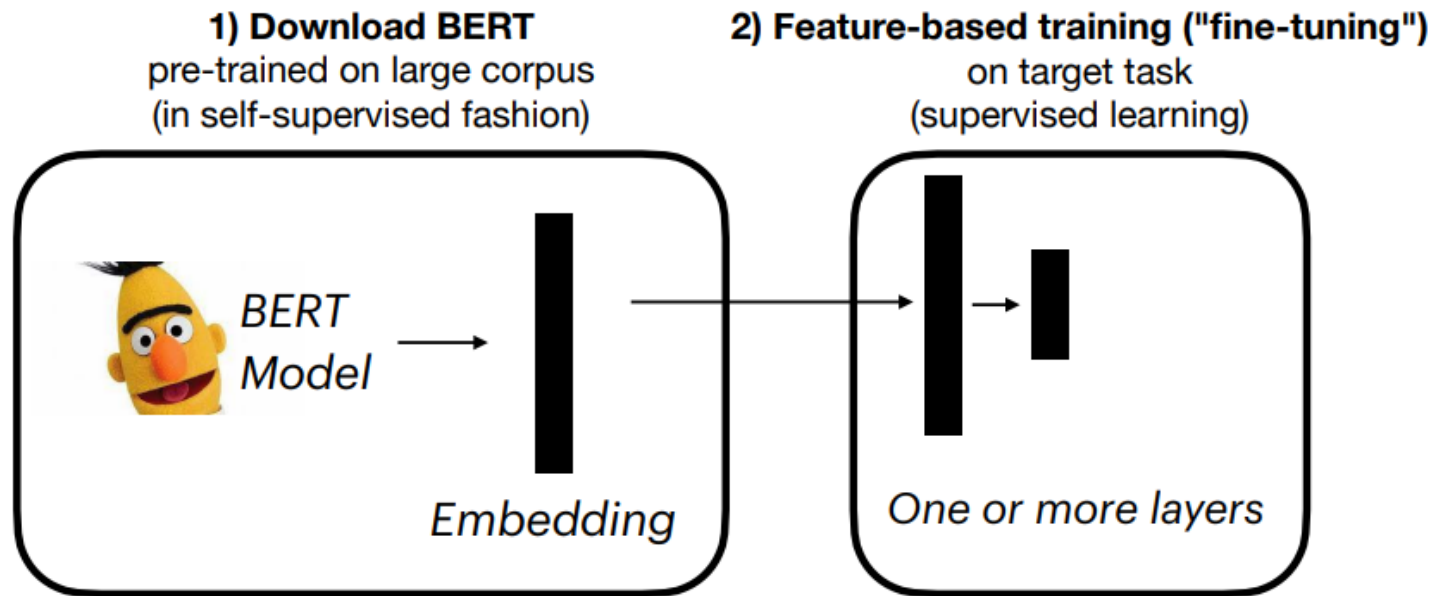
Rendimiento de BERT vs GPT-v1

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

Table 1: GLUE Test results, scored by the evaluation server (<https://gluebenchmark.com/leaderboard>). The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.⁸ BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

BERT preentrenamiento y entrenamiento basado en características

- Mantenga BERT congelado después del entrenamiento previo
- Cree incrustaciones BERT para conjuntos de datos etiquetados para tareas posteriores y entrene un nuevo modelo en estas incrustaciones (en el papel original, biLSTM de 2 capas en incrustaciones de las últimas 4 capas concatenadas se desempeñó mejor)



System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	93.1
Fine-tuning approach		
BERT _{LARGE}	96.6	92.8
BERT _{BASE}	96.4	92.4
Feature-based approach (BERT _{BASE})		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

Algunos modelos de transformadores populares: BERT, GPT y BART

**GPT-v2: Los modelos de lenguaje son aprendices
multitarea sin supervisión**

GPT-v1: Transformador generativo pre-entrenado

- Desarrollado por OpenAI
- Unidireccional: entrenado para predecir la siguiente palabra en una oración

GPT (110 million parameters)

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2 (1.5 billion parameters)

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3 (175 billion parameters)

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. <https://arxiv.org/abs/2005.14165>

Arquitectura de GPT-v1 y tareas posteriores

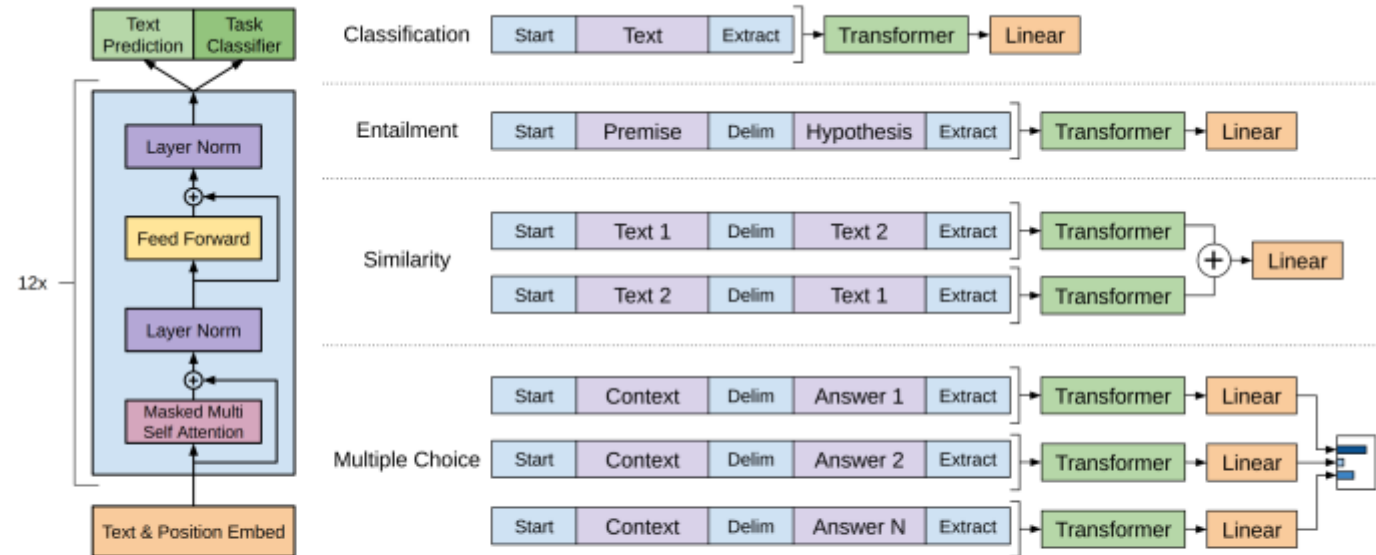


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Conceptos clave de GPT-v2

- Unidireccional como GPT-v1
- Comparado con GPT-v1
 - Modelo más grande (cuanto más grande mejor)
 - Conjunto de datos sin etiquetas más grande (cuanto más grande, mejor)
 - Sin ajuste fino (use transferencia de disparo cero en su lugar)

Arquitectura GPT-v2

- En general, similar a GPT-v1 (que se basa en el decodificador Transformador original)
- Alguna pequeña reordenación de las capas de normal y residuales
- Aumentar el tamaño del vocabulario de 30.000 -> 50.257
- Aumentar el tamaño del contexto de 512 -> 1024 tokens
- En general, 1.5 mil millones en lugar de 110 millones de parámetros


Conjunto de datos de entrenamiento GPT-v2

- WebText (millones de páginas web)
- Calidad del conjunto de datos enfatizada
- Basado en publicaciones de Reddit con más de 3 karma
 - Obtiene 45 millones de enlaces a sitios web
 - Después del preprocesamiento y la limpieza: 8 millones de documentos
 - 40 Gb de texto

Transferencia de tareas Zero-Shot

A diferencia de GPT-v1, no hay instrucciones específicas/reorganización para tareas específicas

<https://huggingface.co/models?filter=zero-shot-classification>



Update: Zero-shot classification is now supported in our API and you can experiment with a number of compatible models on our [Model Hub](#).

Recently, the NLP science community has begun to pay increasing attention to zero-shot and few-shot applications, such as in the [paper from OpenAI](#) introducing GPT-3. This demo shows how 🤗 Transformers can be used for zero-shot topic classification, the task of predicting a topic that the model has not been trained on.

Zero Shot Topic Classification

Choose an example

Custom

Text

What is the color of grass?

Possible topics (separated by ', ')

green,red,blue,nothing,pink,purple 34/1000

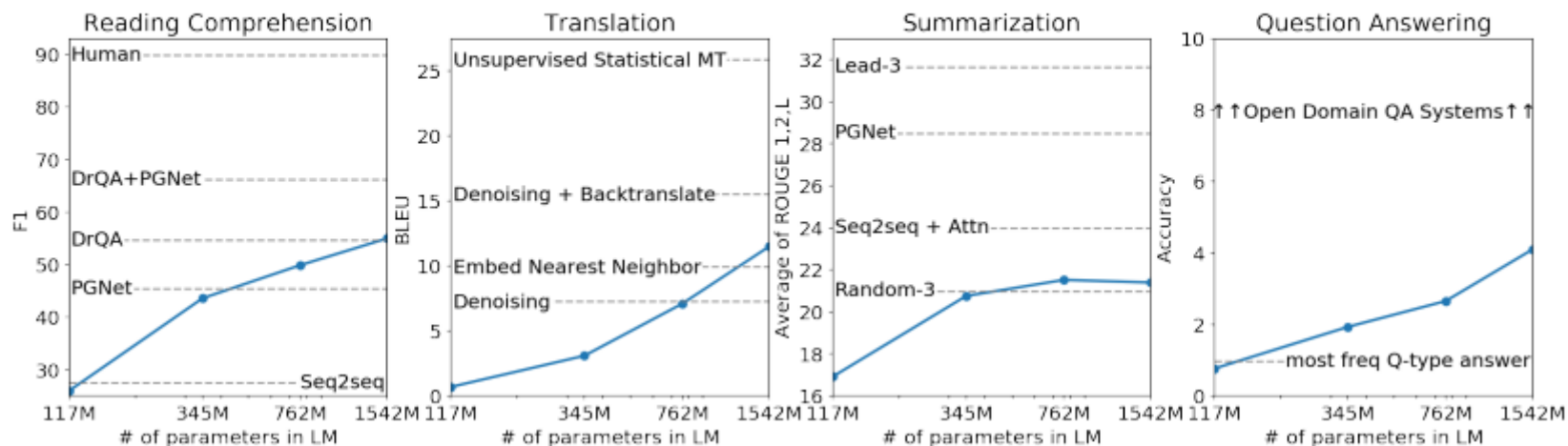
☒ Allow multiple correct topics

Top Predictions

green	72.9%
red	1.6%
nothing	0.6%
blue	0.5%
purple	0.3%

Arquitectura de GPT-v1 y tareas posteriores

Language Models are Unsupervised Multitask Learners



Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW* (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	21.8
117M	35.13	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50	75.20
345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	1.06	26.37	55.72
762M	10.87	60.12	93.45	88.0	19.93	40.31	0.97	1.02	22.05	44.575
1542M	8.63	63.24	93.30	89.05	18.34	35.76	0.93	0.98	17.48	42.16

*mal rendimiento de 1BW probablemente debido a la reorganización a nivel de oración en ese conjunto de datos, por lo que se pierden contextos más grandes y de largo alcance

Algunos modelos de transformadores populares:
BERT, GPT y BART

GPT-v3: Los modelos de lenguaje son aprendices con pocas posibilidades

GPT: (Generative Pre-trained Transformer)

- Desarrollado por OpenAI
- Unidireccional: entrenado para predecir la siguiente palabra en una oración

GPT (110 million parameters)

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

GPT-2 (1.5 billion parameters)

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.
https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

GPT-3 (175 billion parameters)

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. <https://arxiv.org/abs/2005.14165>

Arquitectura GPT-v3

- En general, similar a GPT-v2
- 175 mil millones en lugar de 1.5 mil millones de parámetros (más capas, etc.)
- Duplica el tamaño del contexto (2048 en lugar de 1024)
- Incrustaciones de palabras más grandes (12,8 k en lugar de 1,6 k)
- Patrón de atención de Transformador Escaso*

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences With Sparse Transformers, 2019.

Conjuntos de datos de entrenamiento de GPT-v3

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. “Weight in training mix” refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

Aprendizaje implícito de tareas (...mientras aprende a predecir la siguiente palabra)

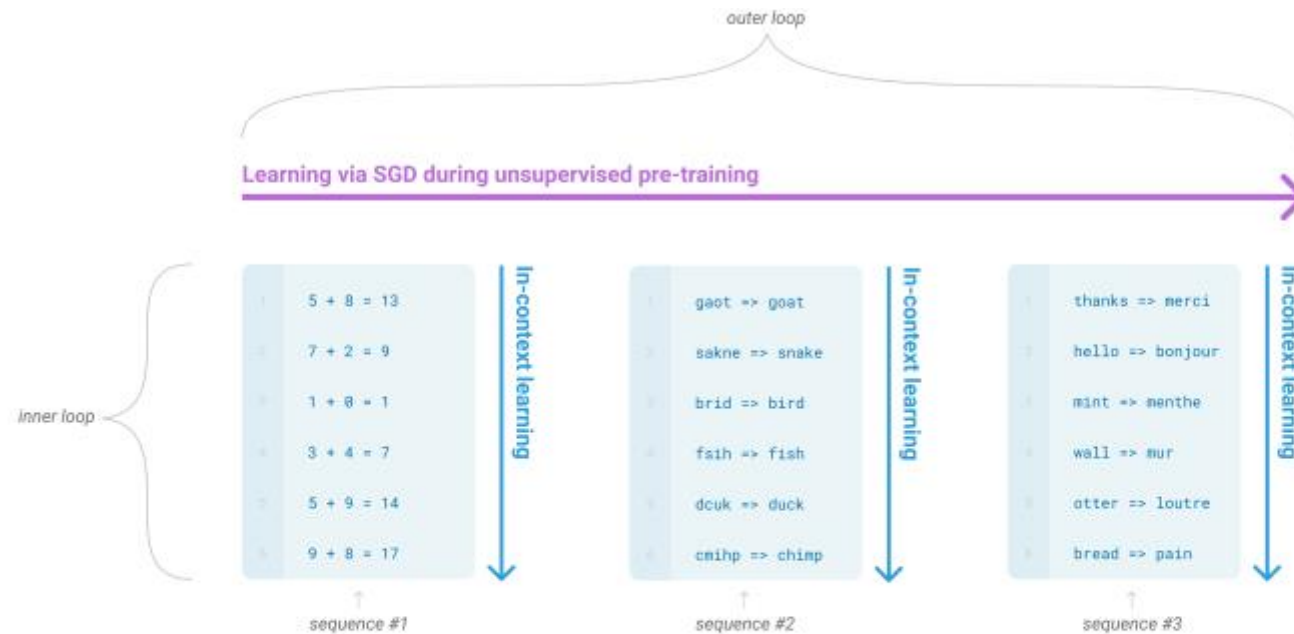


Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term “in-context learning” to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

Mostrar ejemplos vs ajustes precisos

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 cheese => ..... ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← example
3 cheese => ..... ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← examples
3 peppermint => menthe poivrée ←
4 plush giraffe => girafe peluche ←
5 cheese => ..... ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
1 sea otter => loutre de mer ← example #1
↓
gradient update
↓
1 peppermint => menthe poivrée ← example #2
↓
gradient update
↓
...
↓
1 plush giraffe => girafe peluche ← example #N
↓
gradient update
↓
1 cheese => ..... ← prompt
```

Figure 2.1: Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show four methods for performing a task with a language model – fine-tuning is the traditional method, whereas zero-, one-, and few-shot, which we study in this work, require the model to perform the task with only forward passes at test time. We typically present the model with a few dozen examples in the few shot setting. Exact phrasings for all task descriptions, examples and prompts can be found in Appendix G.

Algunos de los muchos resultados...

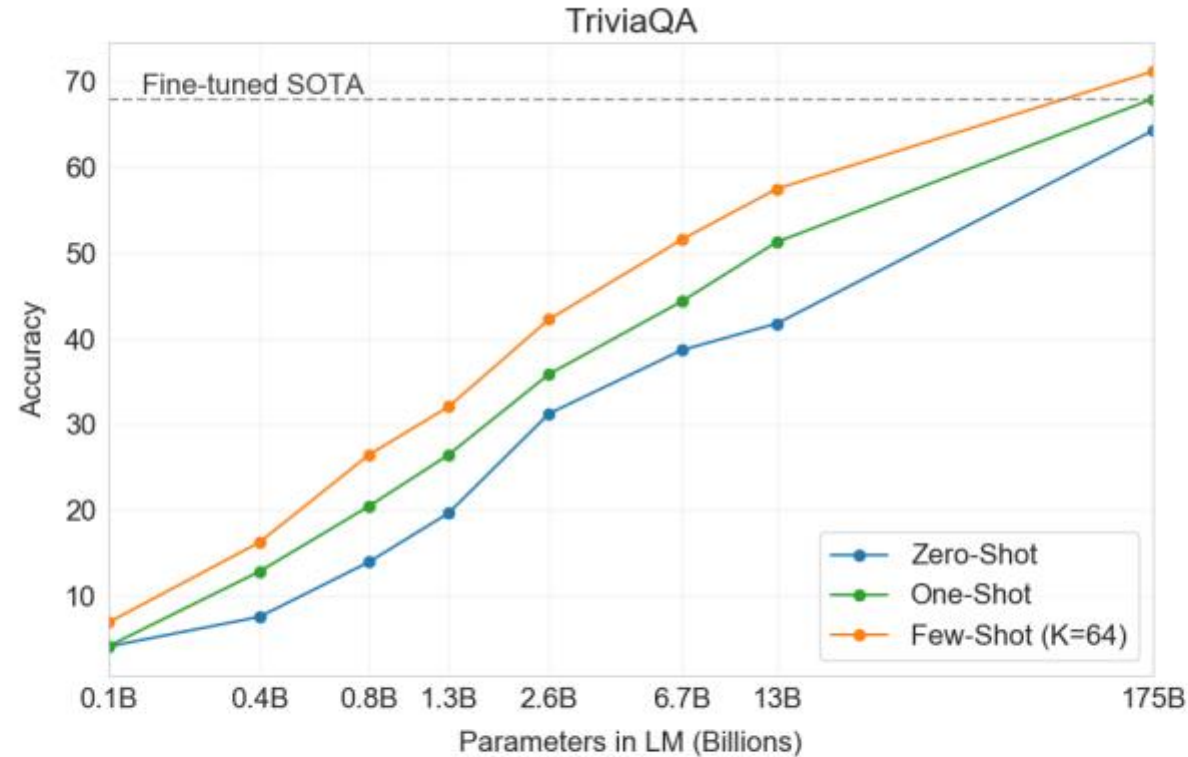


Figure 3.3: On TriviaQA GPT3's performance grows smoothly with model size, suggesting that language models continue to absorb knowledge as their capacity increases. One-shot and few-shot performance make significant gains over zero-shot behavior, matching and exceeding the performance of the SOTA fine-tuned open-domain model, RAG [LPP⁺20]

Algunos modelos de
transformadores populares:
BERT, GPT y BART

**BART: Combinando transformadores bidireccionales y
autorregresivos**

BART: Combining Bidirectional and Auto-Regressive Transformers

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. <http://arxiv.org/abs/1910.13461>

BART de Facebook AI combina BERT de Google y GPT de OpenAI

La naturaleza bidireccional y autocodificadora de BERT es...

+Bueno para tareas posteriores (por ejemplo, clasificación) que requieren información sobre toda la secuencia

-No es tan bueno para tareas de generación donde la palabra generada solo debería depender de palabras generadas previamente

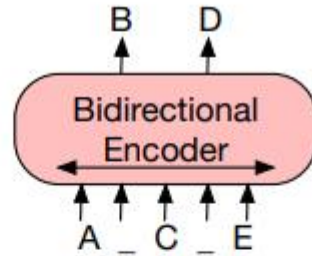
El enfoque unidireccional y autorregresivo de GPT es...

+Bueno para generación de texto

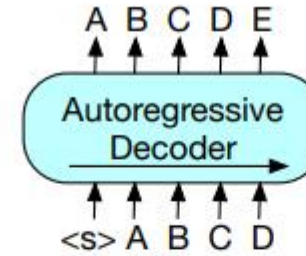
-No tan bueno para tareas que requieren información de secuencias completas, por ejemplo, clasificación

BART es lo mejor de ambos mundos

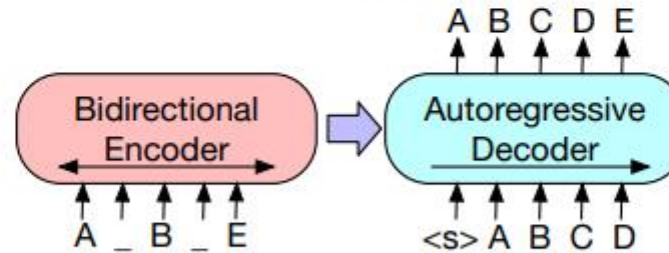
BART: Codificador BERT + Decodificador GPT + Transformaciones de ruido



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitrary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

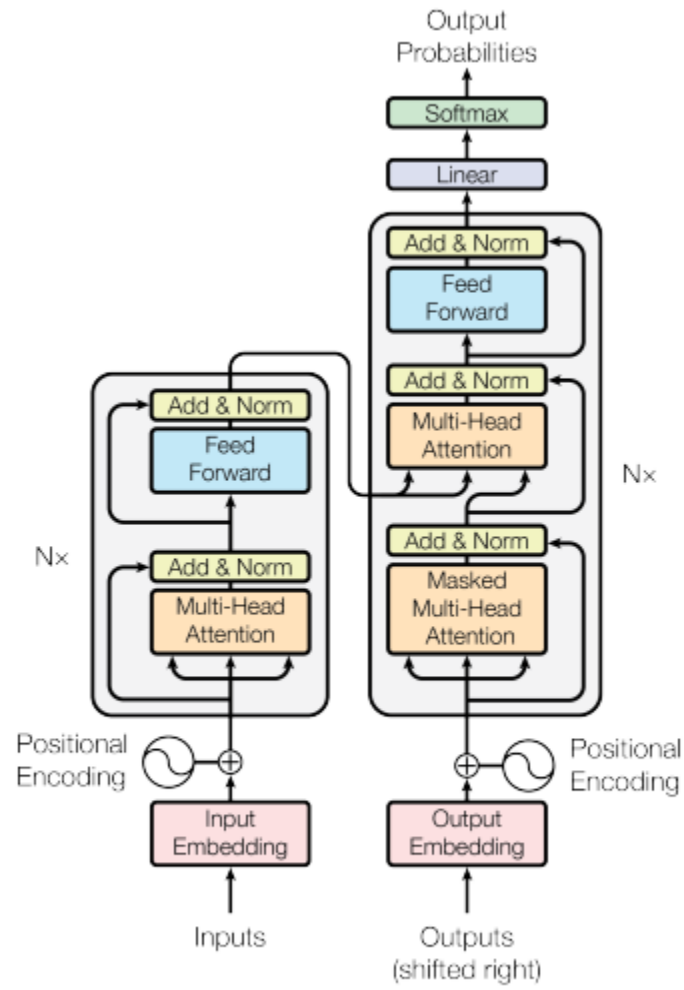


Figure 1: The Transformer - model architecture.

Transformaciones de ruido en BART para el entrenamiento previo sobre datos sin etiquetar

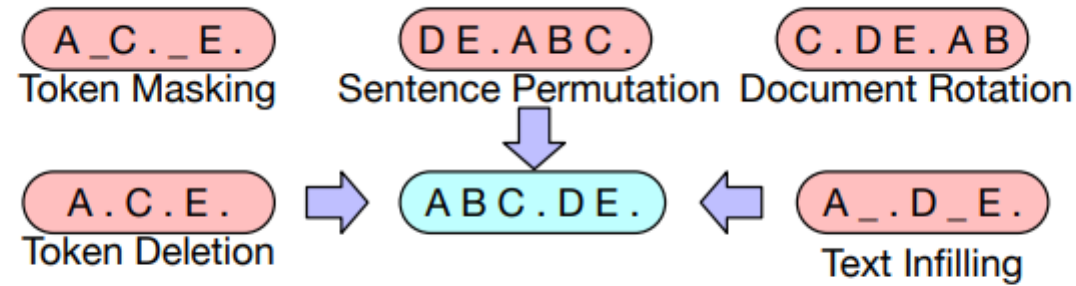


Figure 2: Transformations for noising the input that we experiment with. These transformations can be composed.

Como un codificador automático de eliminación de ruido, optimiza la pérdida de reconstrucción

Rendimiento de BART bajo diferentes transformaciones de ruido

Model	SQuAD 1.1 F1	MNLI Acc	ELI5 PPL	XSum PPL	ConvAI2 PPL	CNN/DM PPL
BERT Base (Devlin et al., 2019)	88.5	84.3	-	-	-	-
Masked Language Model	90.0	83.5	24.77	7.87	12.59	7.06
Masked Seq2seq Language Model	87.0	82.1	23.40	6.80	11.43	6.19
Permuted Language Model	76.7	80.1	21.40	7.00	11.51	6.56
Multitask Masked Language Model	89.1	83.7	24.03	7.69	12.23	6.96
	89.2	82.4	23.73	7.50	12.39	6.74
BART Base						
w/ Token Masking	90.4	84.1	25.05	7.08	11.73	6.10
w/ Token Deletion	90.4	84.1	24.61	6.90	11.46	5.87
w/ Text Infilling	90.8	84.0	24.26	6.61	11.05	5.83
w/ Document Rotation	77.2	75.3	53.69	17.14	19.87	10.59
w/ Sentence Shuffling	85.4	81.5	41.87	10.93	16.67	7.89
w/ Text Infilling + Sentence Shuffling	90.8	83.8	24.17	6.62	11.12	5.41

Ajuste fino de datos etiquetados

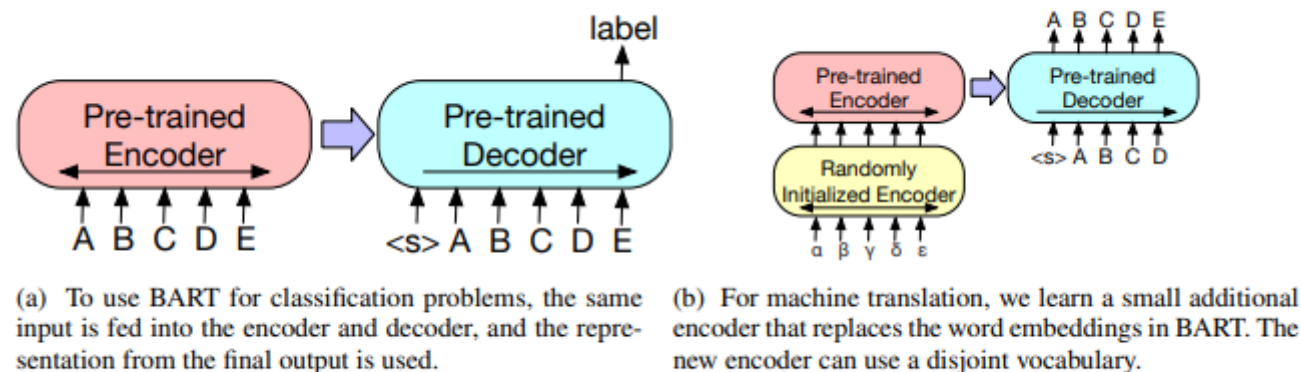


Figure 3: Fine tuning BART for classification and translation.

Rendimiento de BART para tareas discriminatorias

	SQuAD 1.1 EM/F1	SQuAD 2.0 EM/F1	MNLI m/mm	SST Acc	QQP Acc	QNLI Acc	STS-B Acc	RTE Acc	MRPC Acc	CoLA Mcc
BERT	84.1/90.9	79.0/81.8	86.6/-	93.2	91.3	92.3	90.0	70.4	88.0	60.6
UniLM	-/-	80.5/83.4	87.0/85.9	94.5	-	92.7	-	70.9	-	61.1
XLNet	89.0 /94.5	86.1/88.8	89.8/-	95.6	91.8	93.9	91.8	83.8	89.2	63.6
RoBERTa	88.9/ 94.6	86.5/89.4	90.2/90.2	96.4	92.2	94.7	92.4	86.6	90.9	68.0
BART	88.8/ 94.6	86.1/89.2	89.9/90.1	96.6	92.5	94.9	91.2	87.0	90.4	62.8

Table 2: Results for large models on SQuAD and GLUE tasks. BART performs comparably to RoBERTa and XLNet, suggesting that BART’s uni-directional decoder layers do not reduce performance on discriminative tasks.

Rendimiento de BART para tareas generativas

	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	44.16	21.28	40.90	45.14	22.27	37.25

Table 3: Results on two standard summarization datasets. BART outperforms previous work on summarization on two tasks and all metrics, with gains of roughly 6 points on the more abstractive dataset.

	ConvAI2	
	Valid F1	Valid PPL
Seq2Seq + Attention	16.02	35.07
Best System	19.09	17.51
BART	20.72	11.85

Table 4: BART outperforms previous work on conversational response generation. Perplexities are renormalized based on official tokenizer for ConvAI2.

	ELI5		
	R1	R2	RL
Best Extractive	23.5	3.1	17.5
Language Model	27.8	4.7	23.1
Seq2Seq	28.3	5.1	22.8
Seq2Seq Multitask	28.9	5.4	23.1
BART	30.6	6.2	24.3

Table 5: BART achieves state-of-the-art results on the challenging ELI5 abstractive question answering dataset. Comparison models are from Fan et al. (2019).

Algunos modelos de transformadores populares:
BERT, GPT y BART

**Palabras de cierre: El reciente crecimiento de los
transformadores del lenguaje**

Transformadores para secuencias más largas

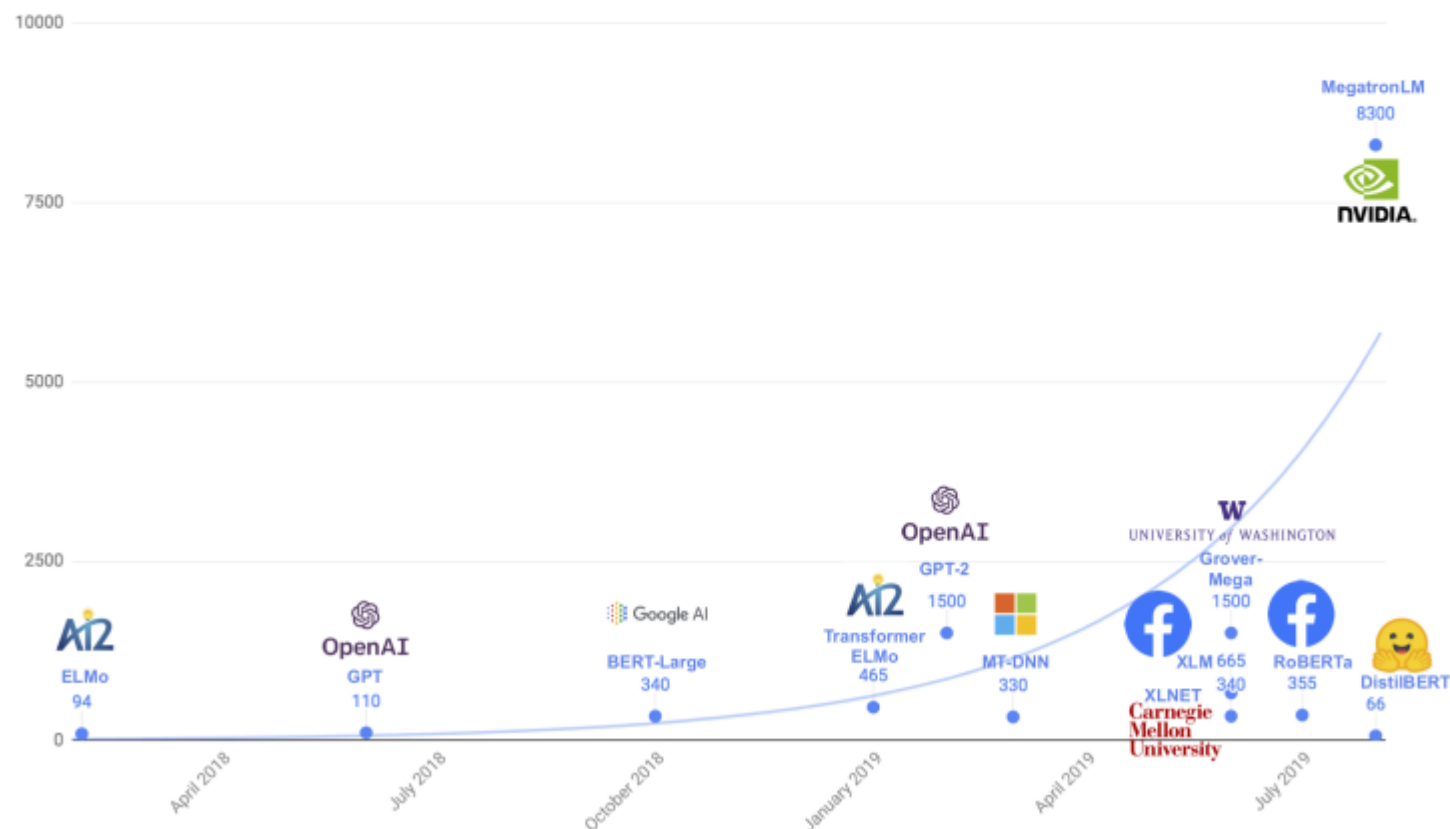
Transformador-XL:

- Modelo sin codificador, solo decodificador
- Entrenado para predecir la siguiente palabra en una oración
- Utiliza estados ocultos para recordar el segmento de texto anterior (512 token)
- "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context", Dai et al. 2019. <https://arxiv.org/abs/1901.02860>

Longformer:

- En lugar de un mecanismo de atención que escala cuadráticamente, utiliza un mecanismo de atención que escala linealmente con la longitud de la secuencia
- Utiliza segmentos de texto extremadamente largos (miles de tokens); similar a RoBERTa
- "Longformer: The Long-Document Transformer", Beltagy et al. 2020. <https://arxiv.org/abs/2004.05150>

GPT-3 (175 billion)



Fuente imagen: <https://medium.com/huggingface/distilbert-8cf3380435b5>

TECH / ARTIFICIAL INTELLIGENCE

OpenAI's text-generating system GPT-3 is now spewing out 4.5 billion words a day

Robot-generated writing looks set to be the next big thing

By **James Vincent** | Mar 29, 2021, 8:24am EDT

<https://www.theverge.com/2021/3/29/22356180/openai-gpt-3-text-generation-words-day>

THE COST OF TRAINING NLP MODELS

A CONCISE OVERVIEW

Or Sharir
AI21 Labs
ors@ai21.com

Barak Peleg
AI21 Labs
barakp@ai21.com

Yoav Shoham
AI21 Labs
yoavs@ai21.com

April 2020

Costos: No aptos para los débiles de corazón

- \$ 2.5k - \$ 50k (modelo de parámetro de 110 millones)
- \$ 10k - \$ 200k (modelo de parámetros de 340 millones)
- \$ 80k - \$ 1.6m (modelo de parámetros de 1.500 millones)

Transformadores para una mejor eficiencia

Reformer:

- La atención del producto punto se reemplaza con la atención de hash sensible a la localidad (LSH)
- Esto logra la atención con $O(n \log(n))$ en lugar del costo de la memoria $O(n^2)$

Kitaev, N., Kaiser, Ł. and Levskaya, A., 2020. Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451. <https://arxiv.org/abs/2001.04451>

ALBERT:

- Tamaño 5 veces más pequeño que BERT con el mismo rendimiento, debido a la compresión mediante poda

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P. and Soricut, R., 2019.

Albert: A lite BERT for self-supervised learning of language representations. arXiv

- preprint arXiv:1909.11942. <https://arxiv.org/abs/1909.11942>

1. Generación de secuencias con RNN
2. Caracter RNN en PyTorch
3. RNN con atención
4. Atención es todo lo que necesitamos
 - 4.1 Forma básica de autoatención
 - 4.2 Atención personal y atención de productos punto
 - 4.3 Atención multicabezal
5. Modelos de transformadores
 - 5.1 La arquitectura del transformador
 - 5.2 Algunos modelos de transformadores populares: BERT, GPT y BART
6. Transformador en PyTorch