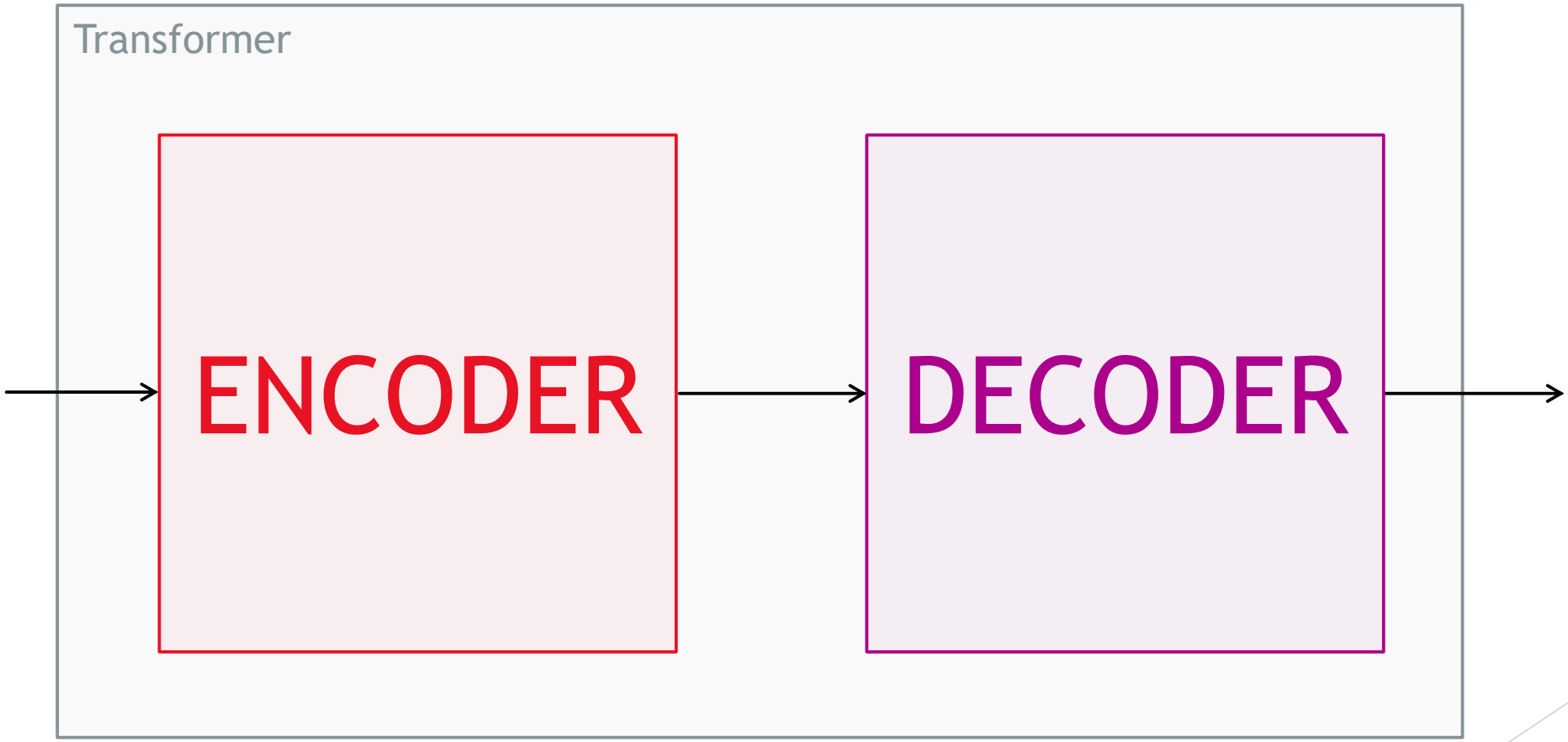
# Attention is all you need (2017)

## How do transformers work?

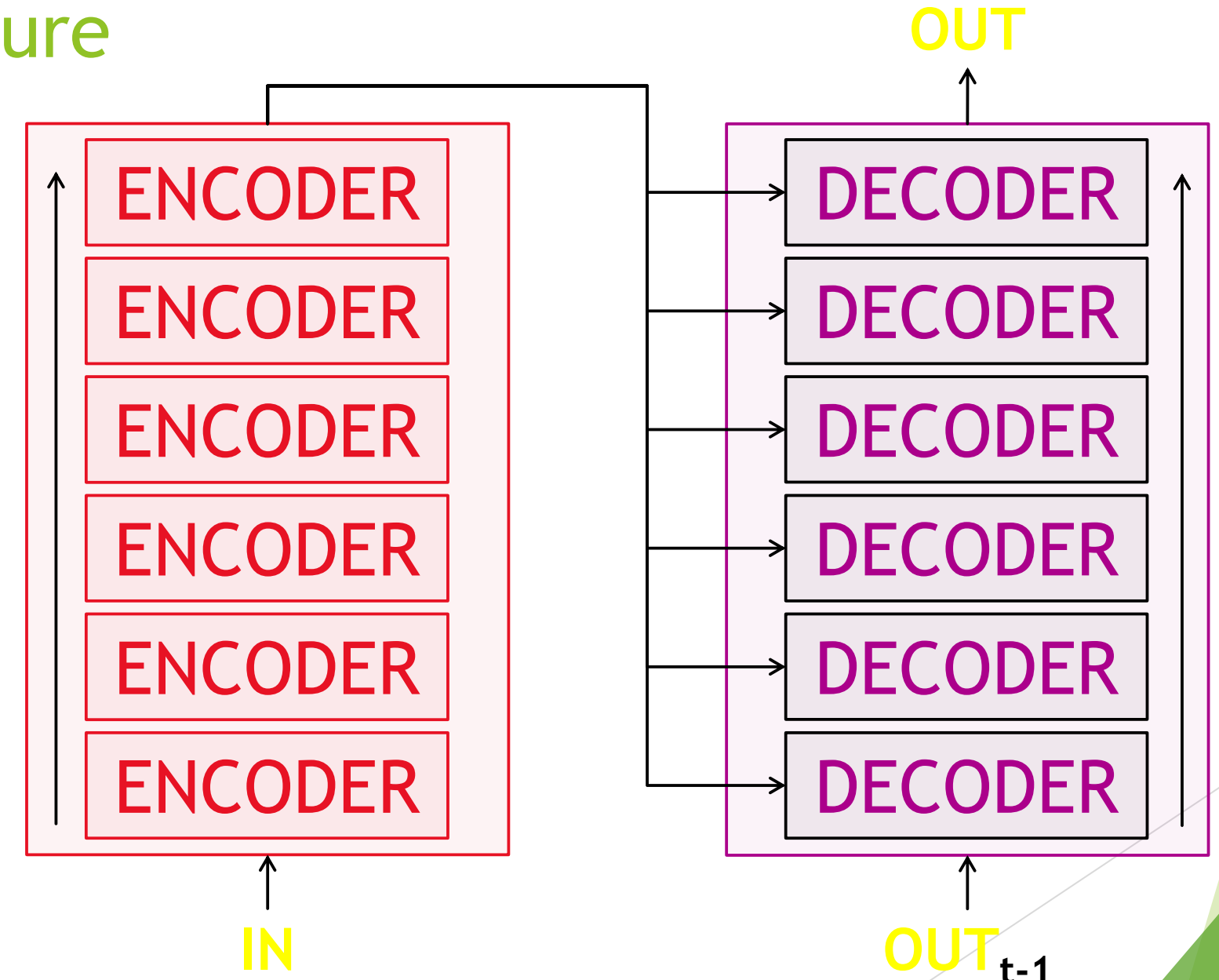
Alejandro C. Parra Garcia

# Translation model

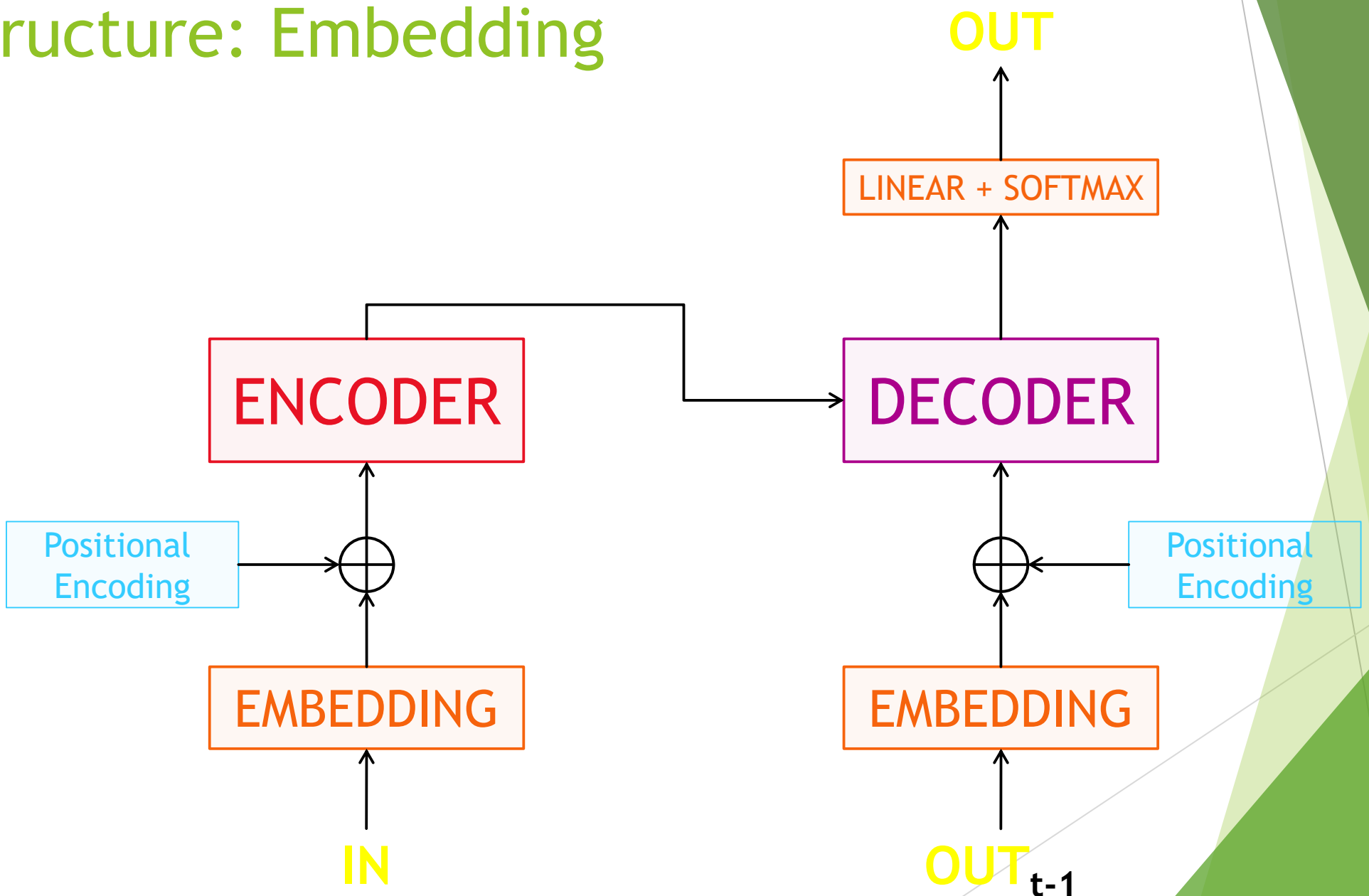
# Structure



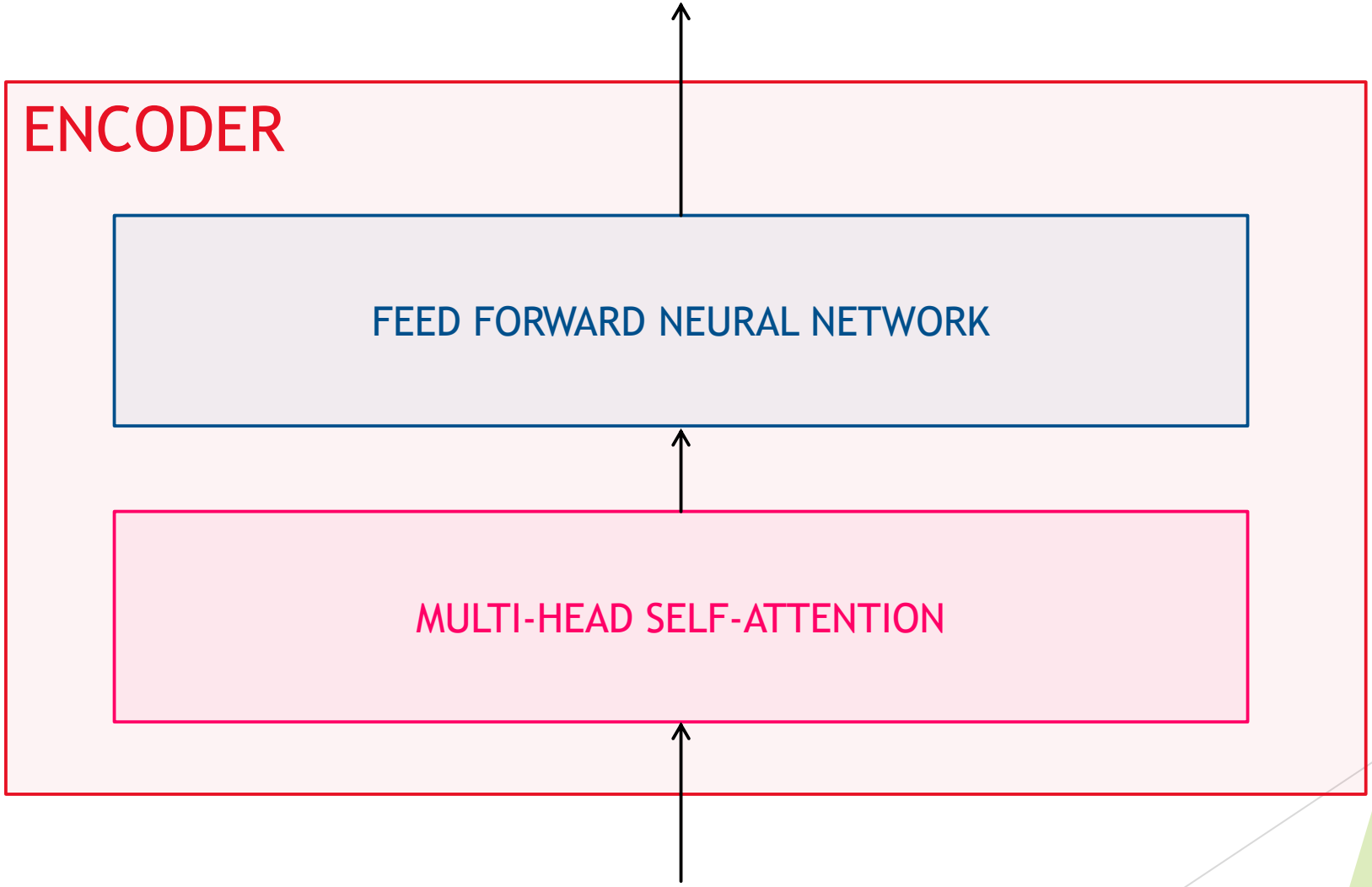
# Structure


OUT
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
IN
OUT t-1

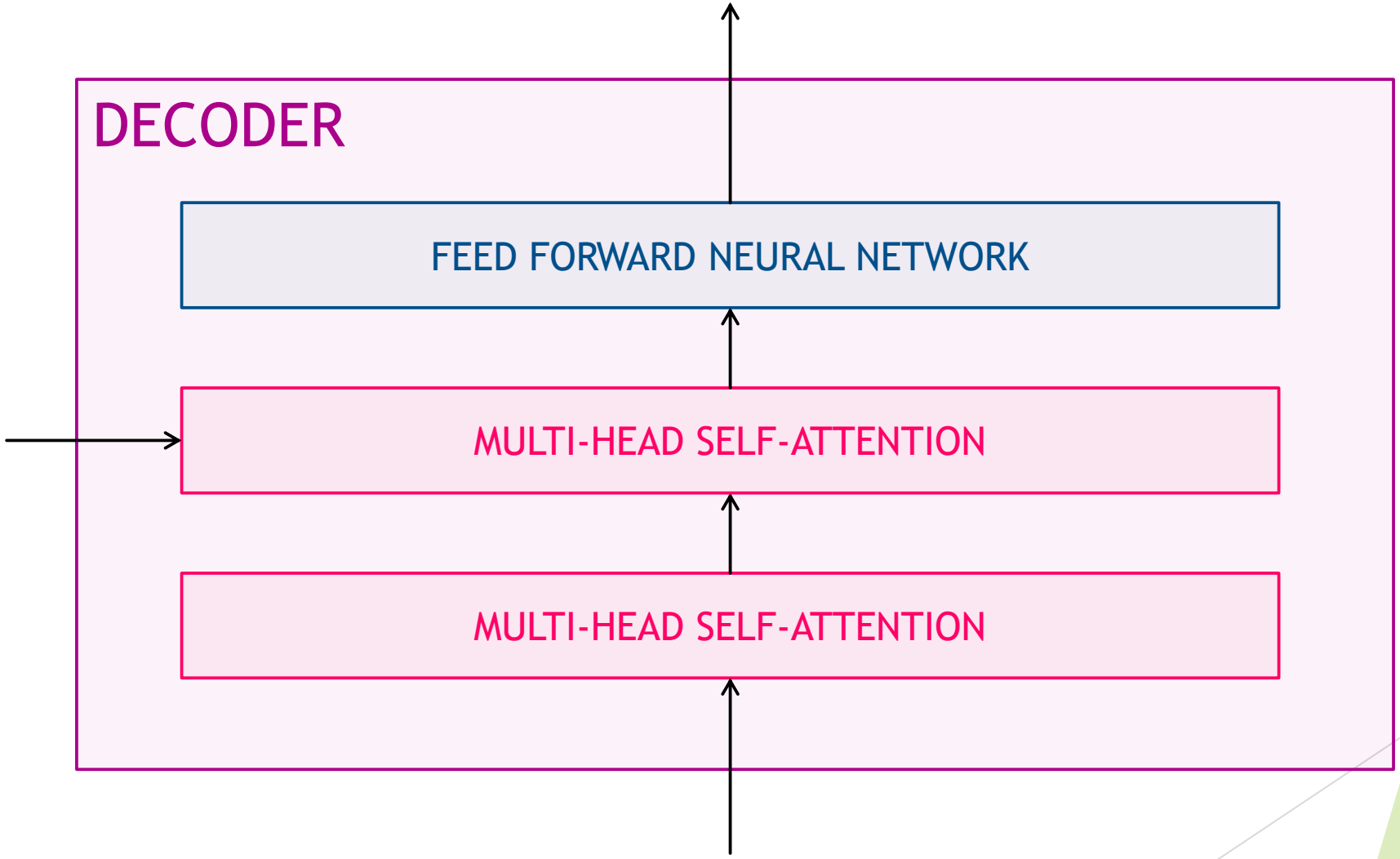
# Structure: Embedding



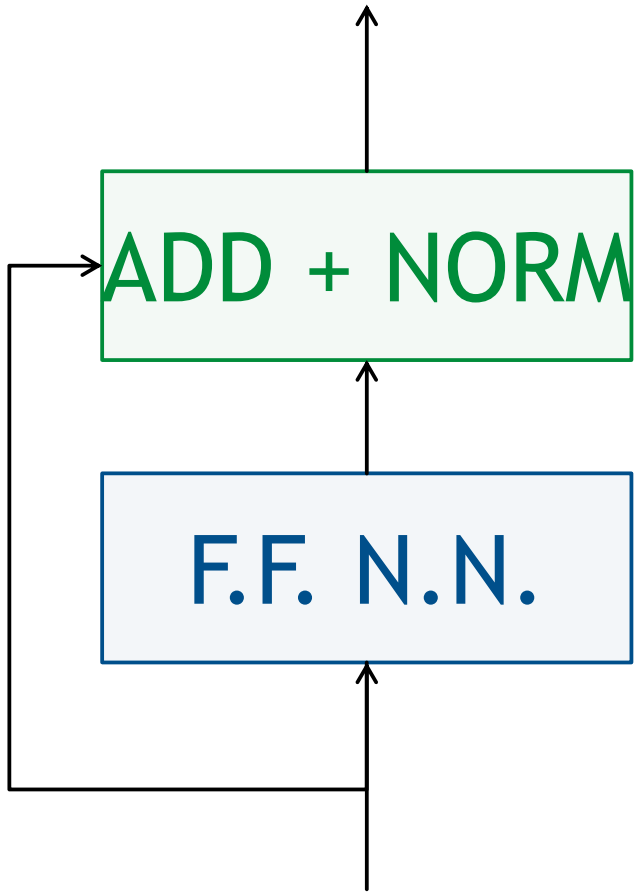
# Structure: Encoder

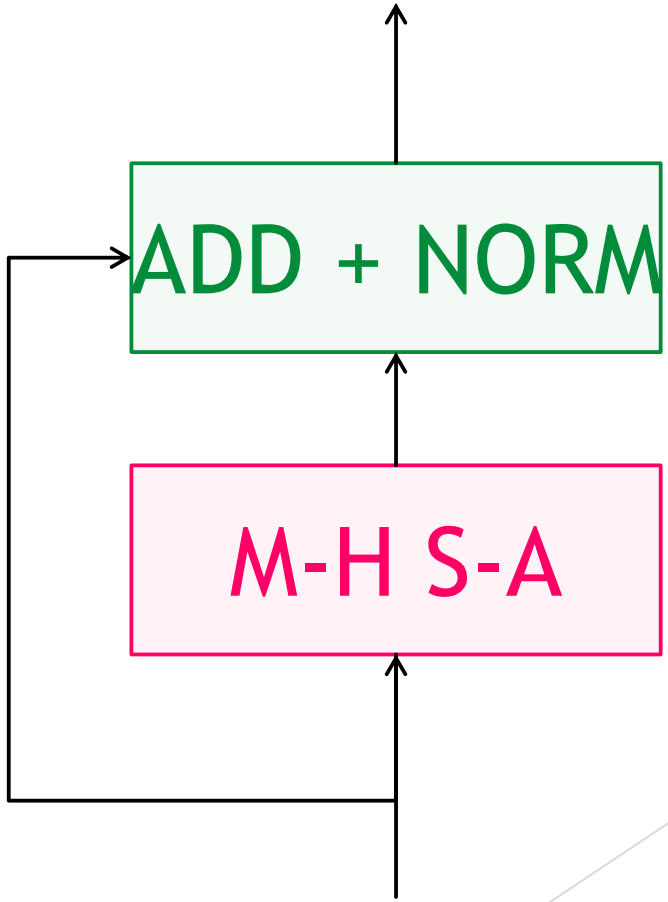


# Structure: Decoder



# Structure: Residuals



# Complete Structure

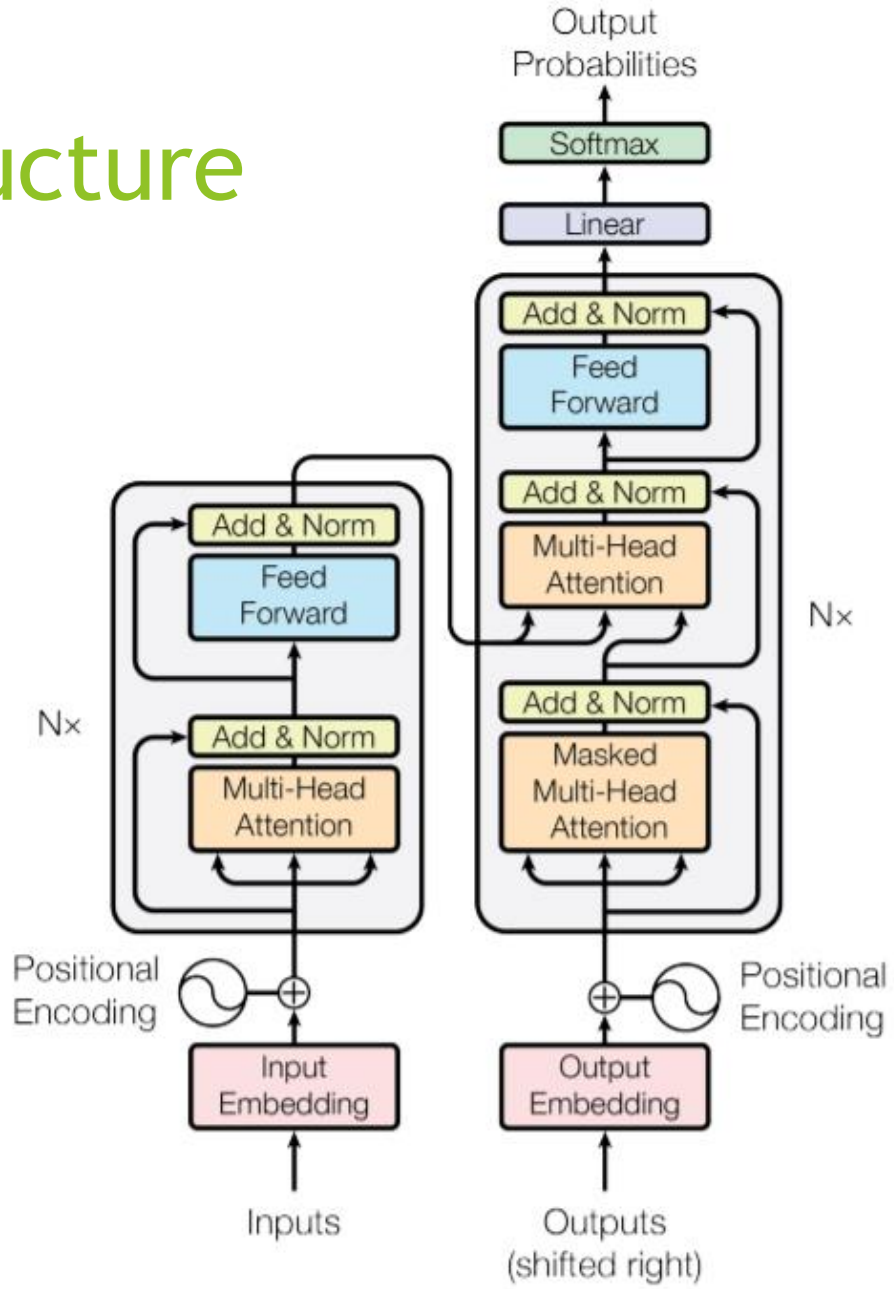



Figure 1: The Transformer - model architecture.

# How it works

# Embedding

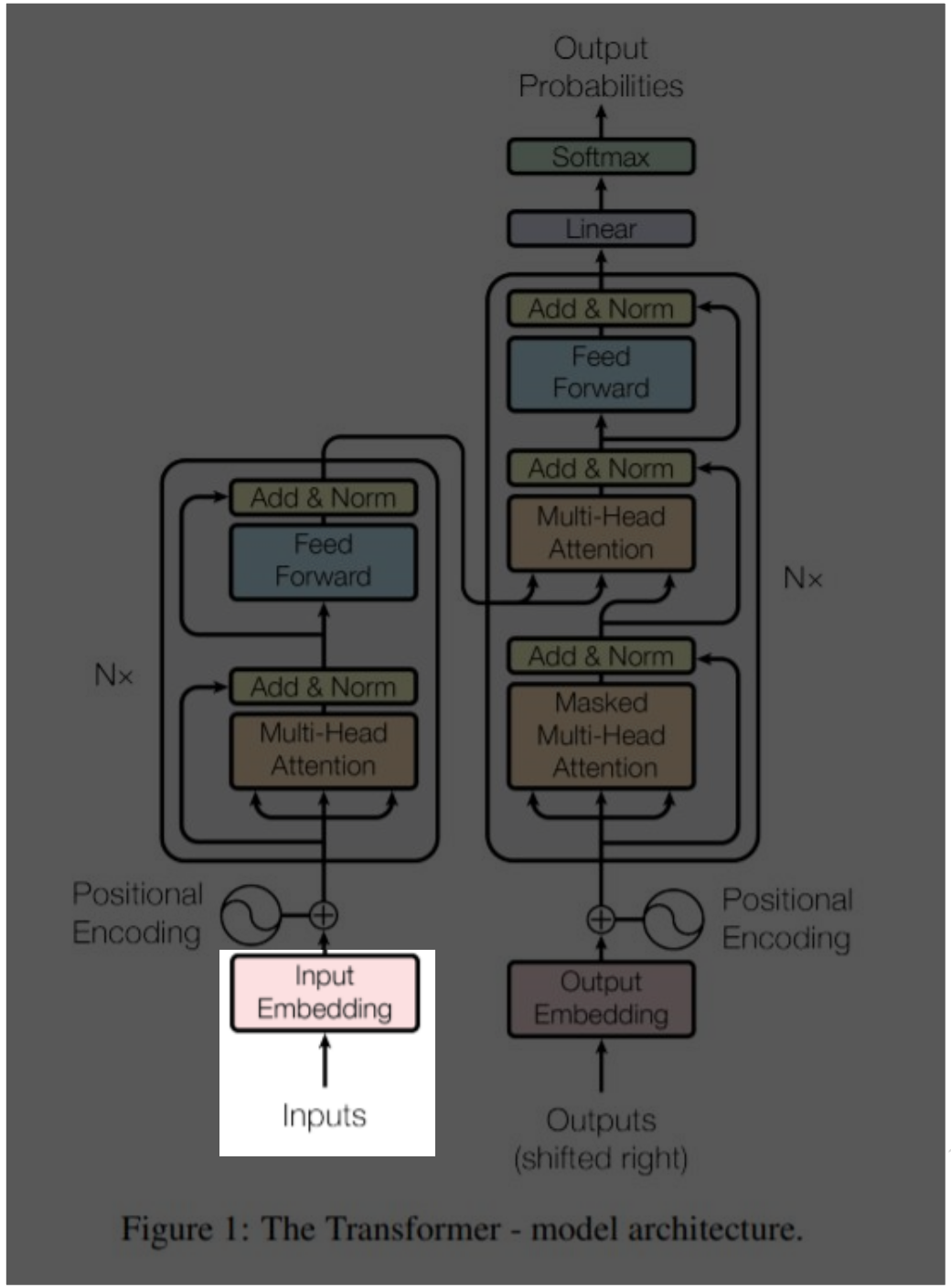



Figure 1: The Transformer - model architecture.

# Embedding

Mi → $x_1$ d=512

nombre → $x_2$ d=512

es → $x_3$ d=512

Alex → $x_4$ d=512

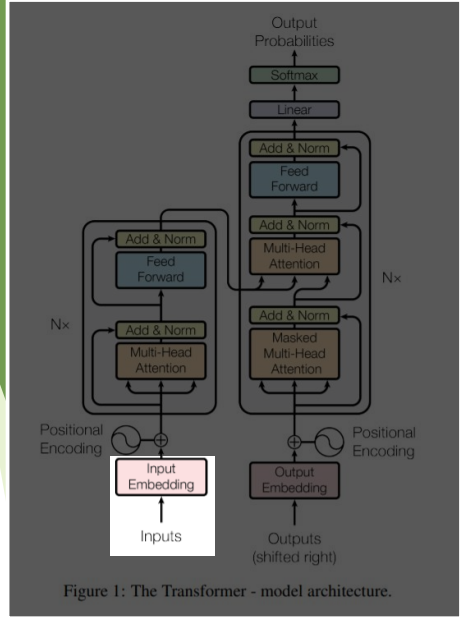


Figure 1: The Transformer - model architecture.

# Positional Enc

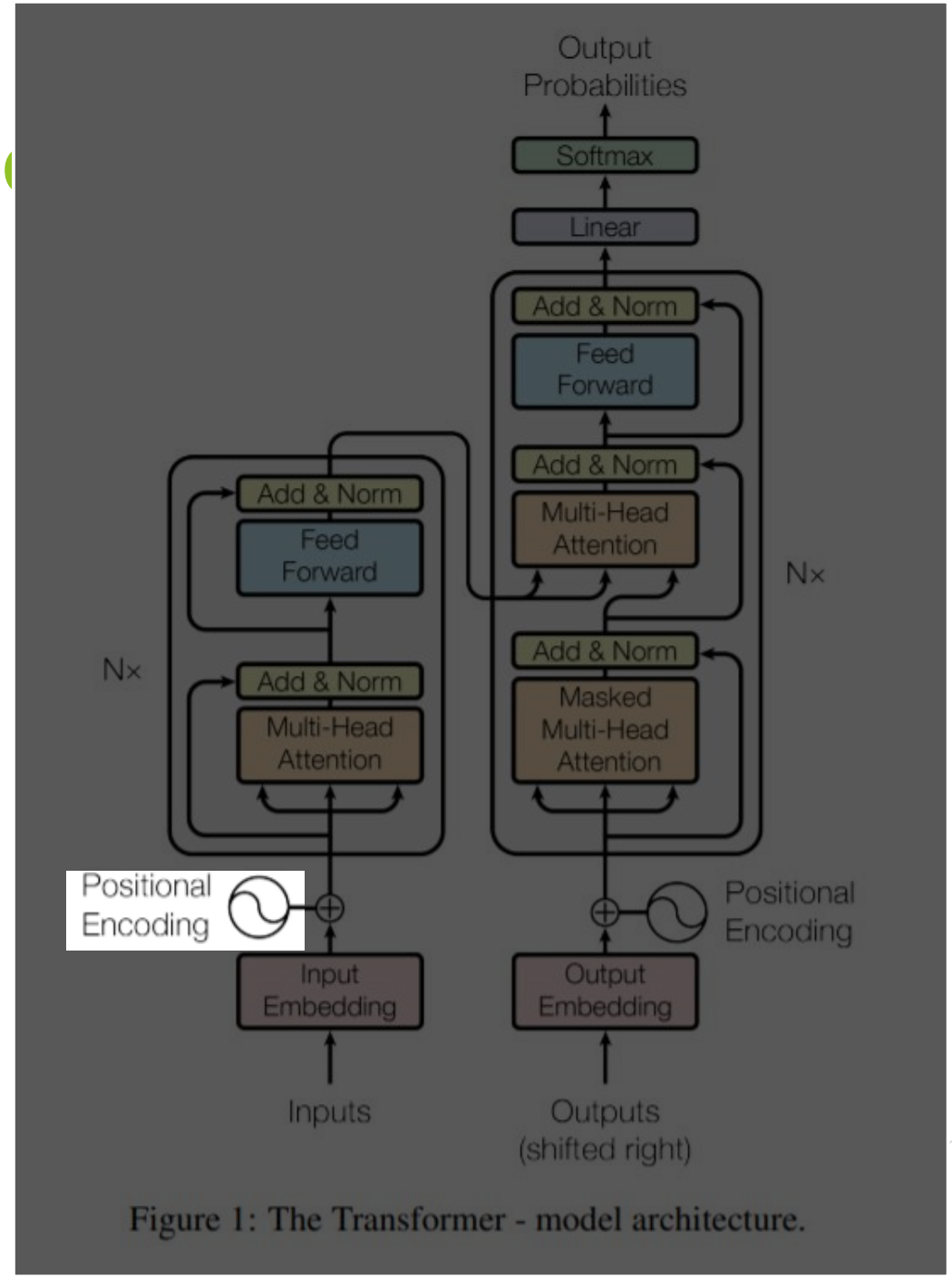



Figure 1: The Transformer - model architecture.

# Positional Encoding

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

where $pos$ is the position and $i$ is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from $2\pi$ to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset $k$, $PE_{pos+k}$ can be represented as a linear function of $PE_{pos}$.

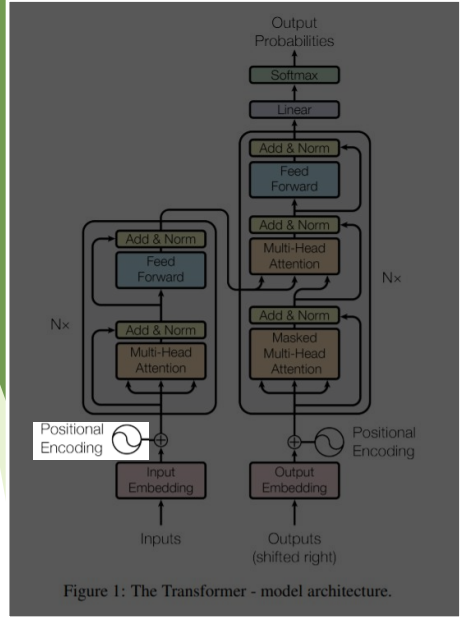


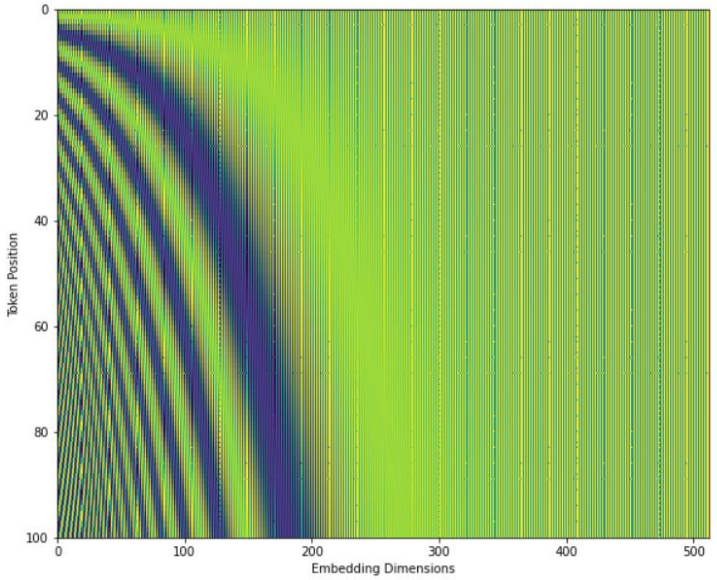
Figure 1: The Transformer - model architecture.

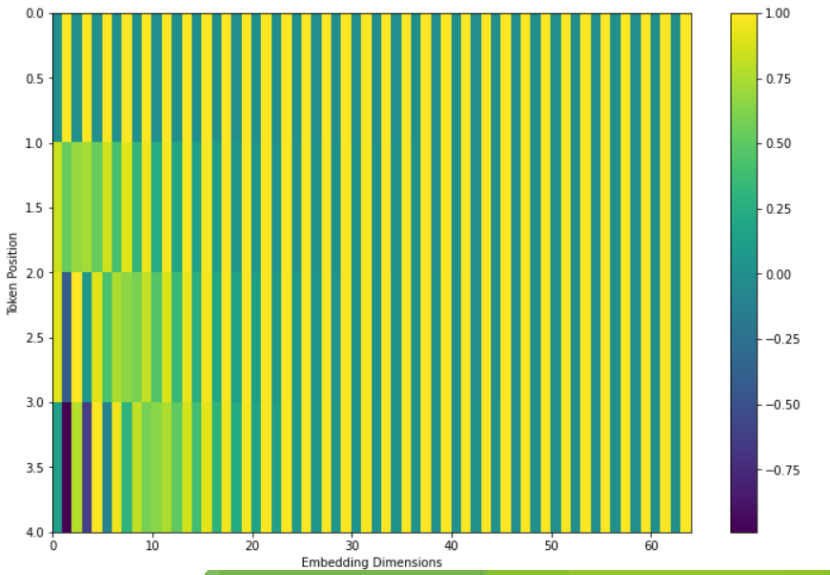
Mi → $Pe_1$ d=512

nombre → $Pe_2$ d=512

es → $Pe_3$ d=512

Alex → $Pe_4$ d=512




https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb

# Embedding + P

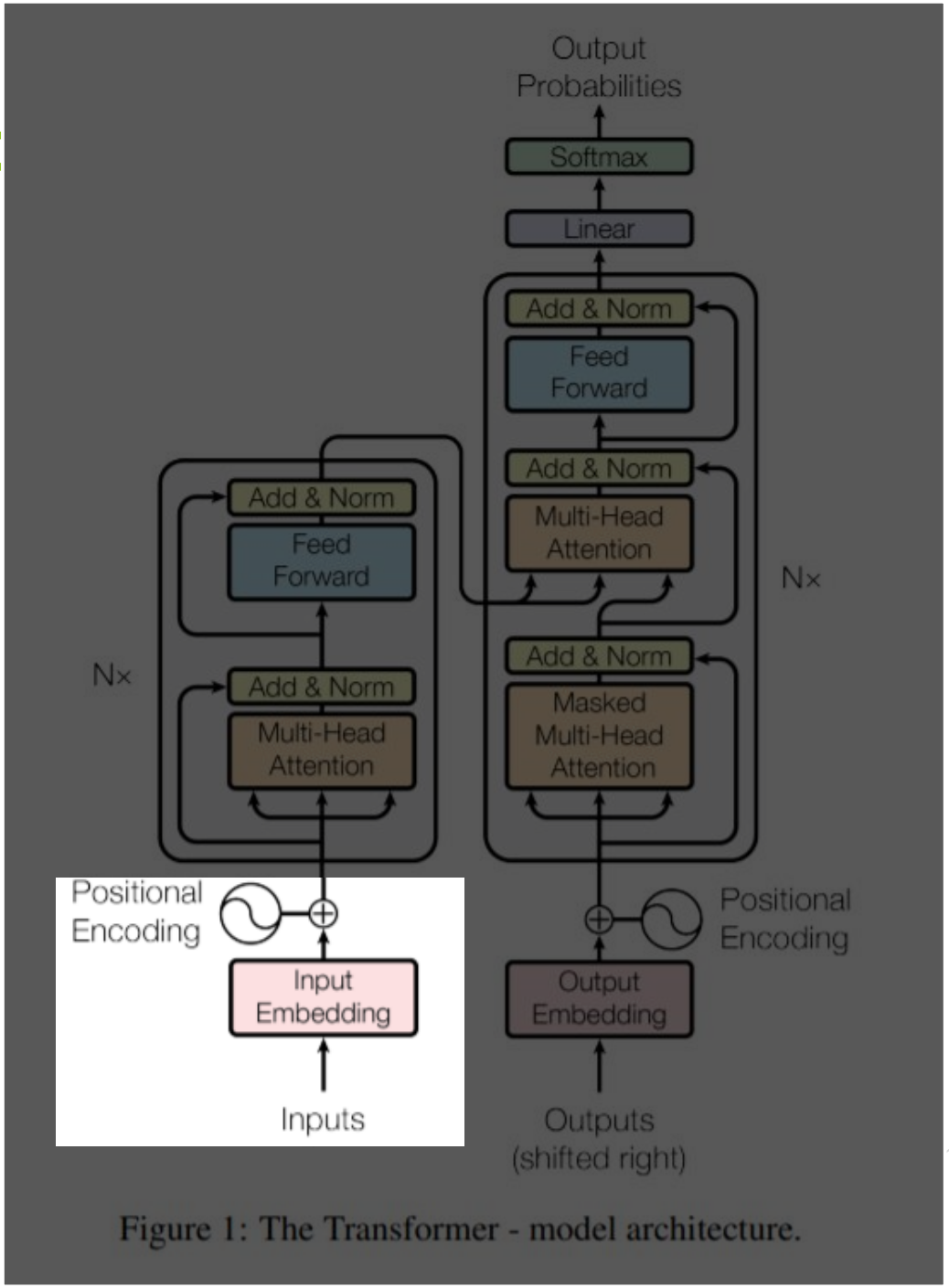



Figure 1: The Transformer - model architecture.

# Embedding + Pos. Encoding

| | | | | | |
|---|---|---|---|---|---|
| $x_1$ | d=512 | | $Pe_1$ | d=512 | $X^p_1$ d=512 |
| $x_2$ | d=512 | | $Pe_2$ | d=512 | $x^p_2$ d=512 |
| $x_3$ | d=512 | + | $Pe_3$ | d=512 | = $x^p_3$ d=512 |
| $x_4$ | d=512 | | $Pe_4$ | d=512 | $x^p_4$ d=512 |

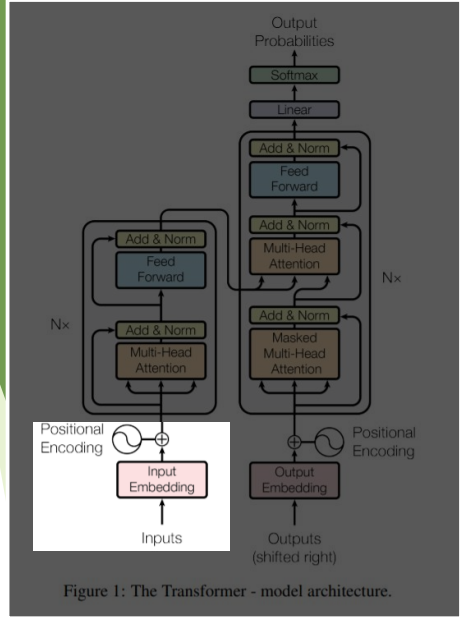


Figure 1: The Transformer - model architecture.

# Multi-Head Sel

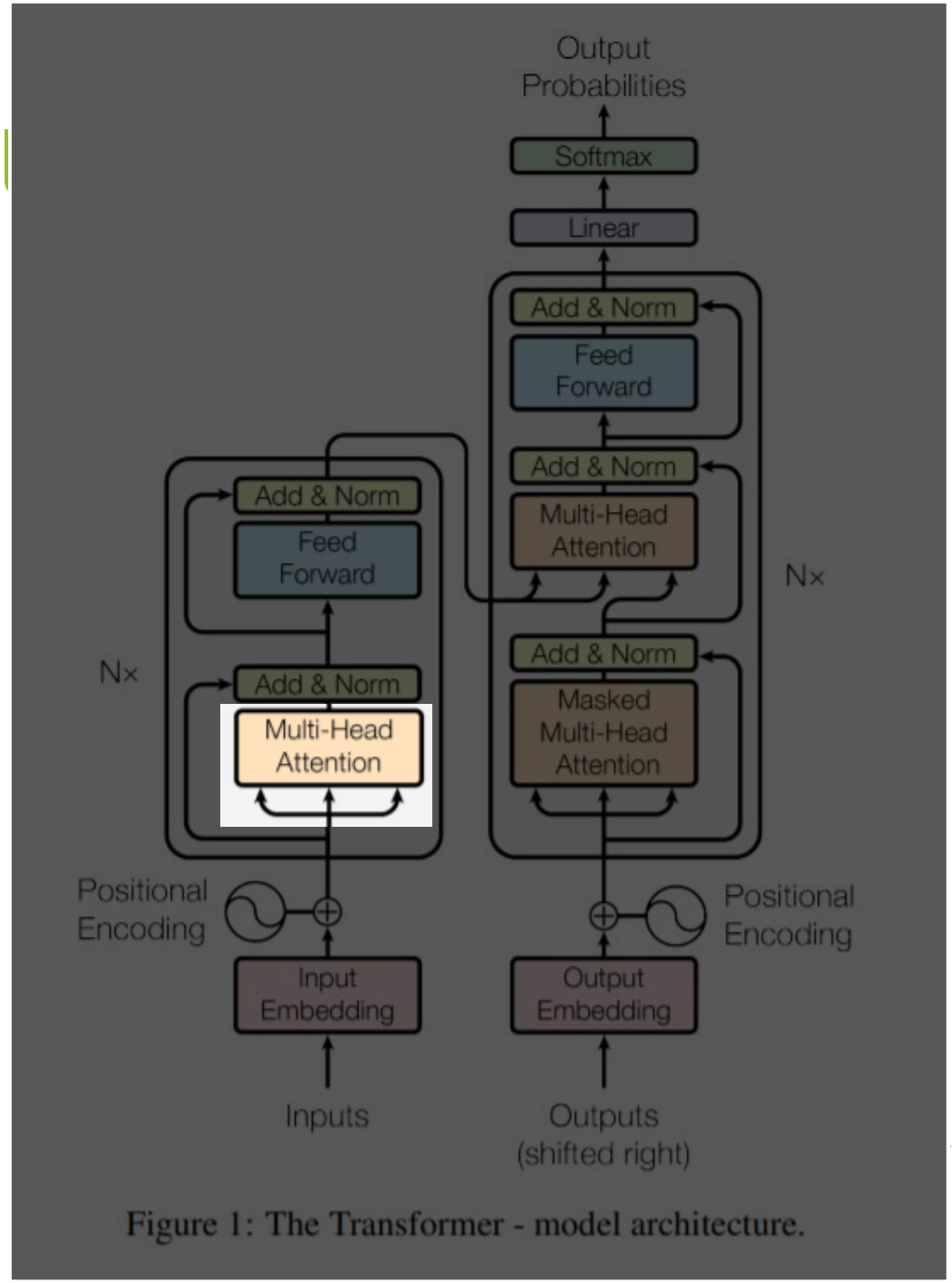
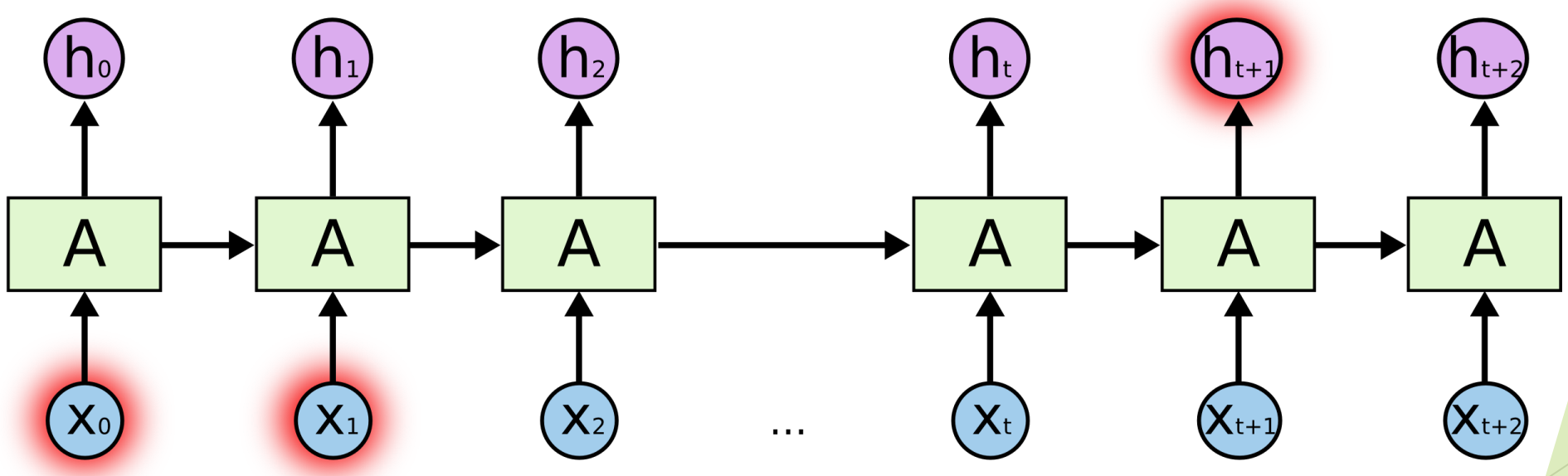



Figure 1: The Transformer - model architecture.

# Multi-Head Self-Attention

## Substitute for RNN

RNN Diagram



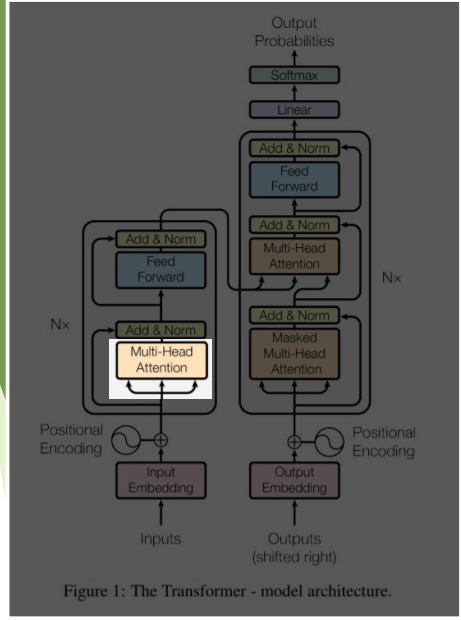




Figure 1: The Transformer - model architecture.

# Multi-Head Self-Attention

## Substitute for RNN

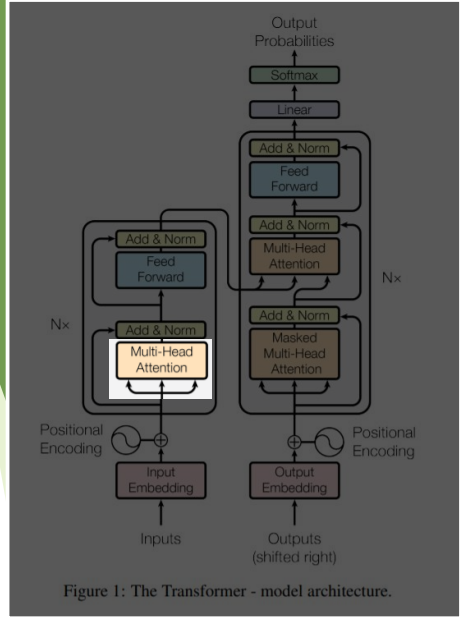
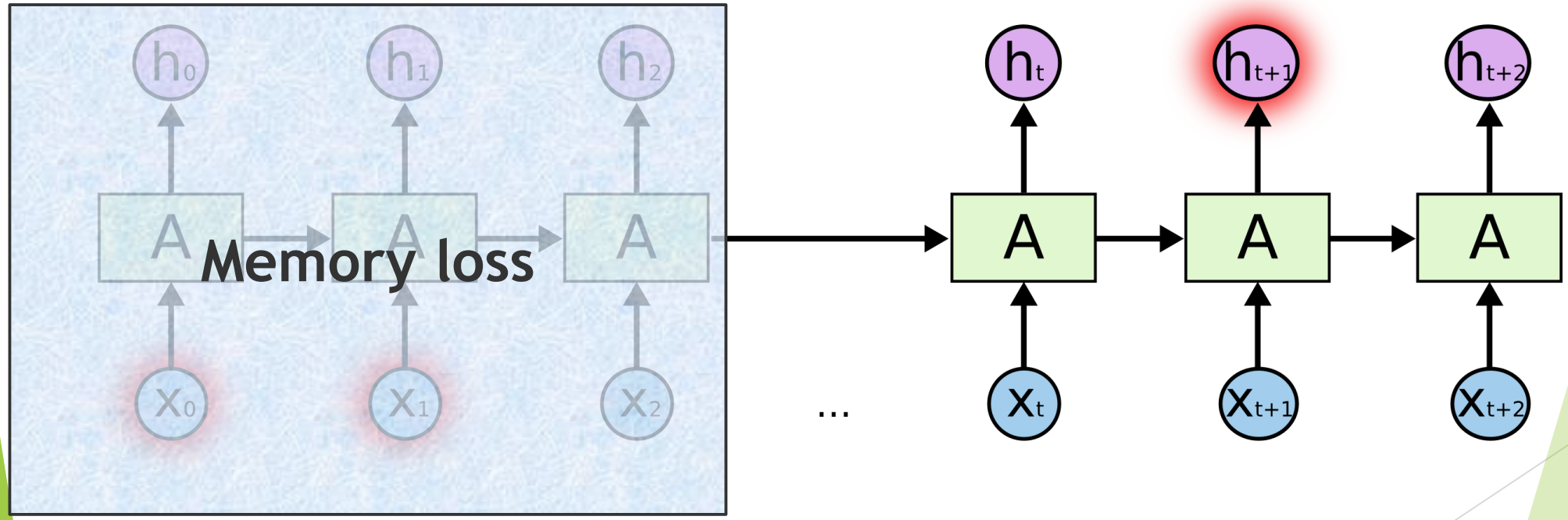


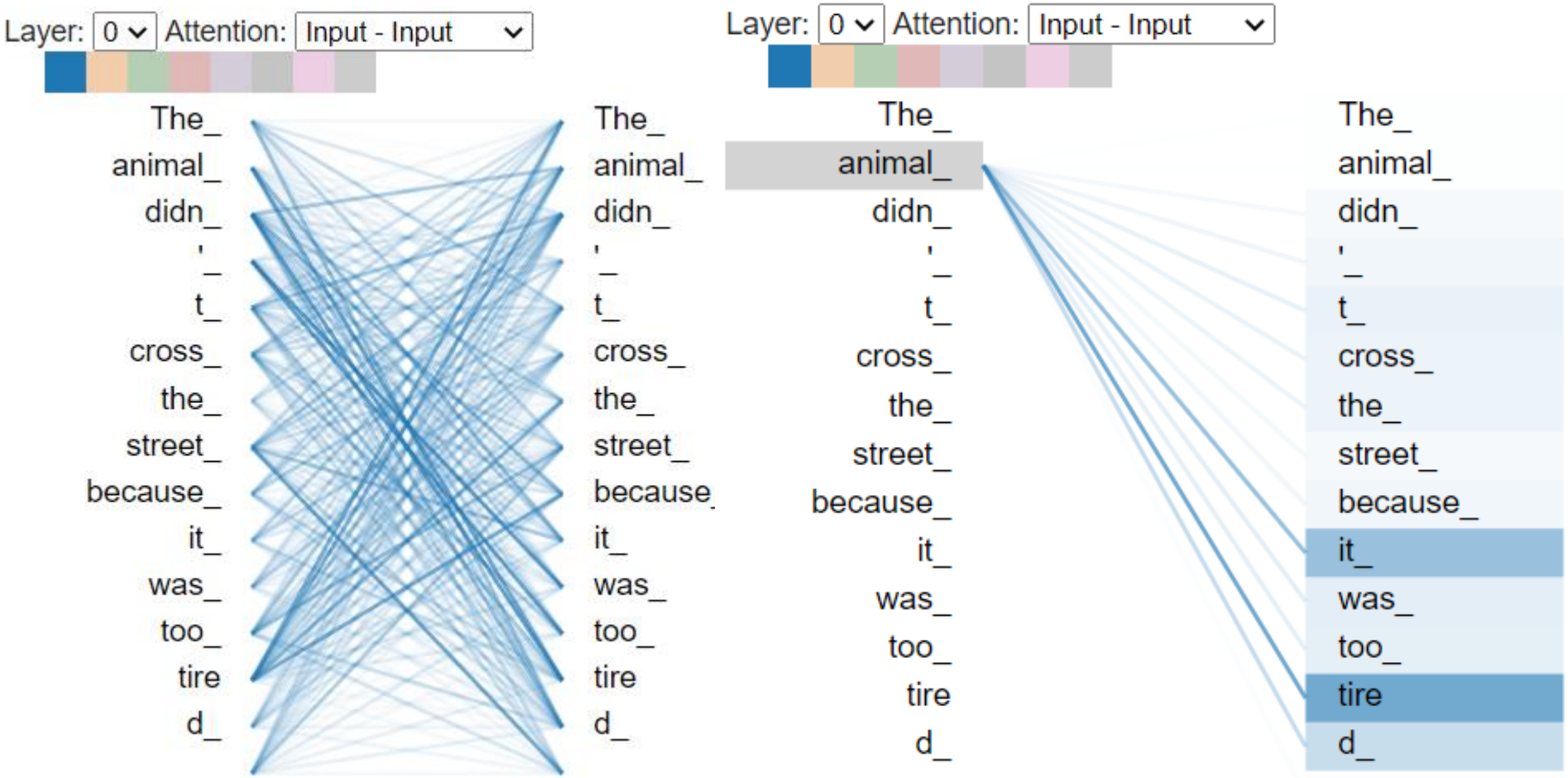
Figure 1: The Transformer - model architecture.

RNN problem



# Multi-Head Self-Attention

## Substitute for RNN







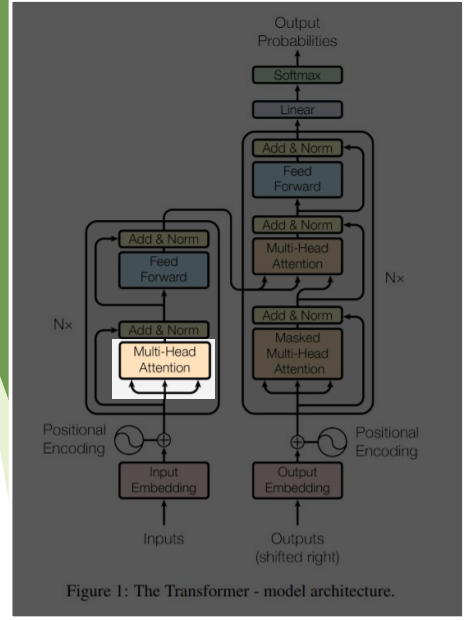


Figure 1: The Transformer - model architecture.

https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

# Multi-Head Self-Attention

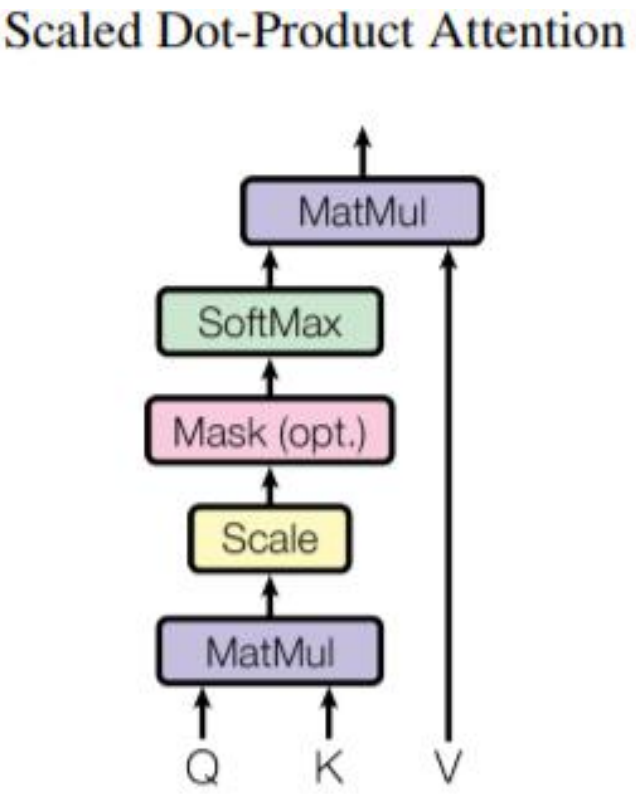
## Self-Attention

The sublayers needs 3 vectors for each word (d=64)

Queries

Keys

Values




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

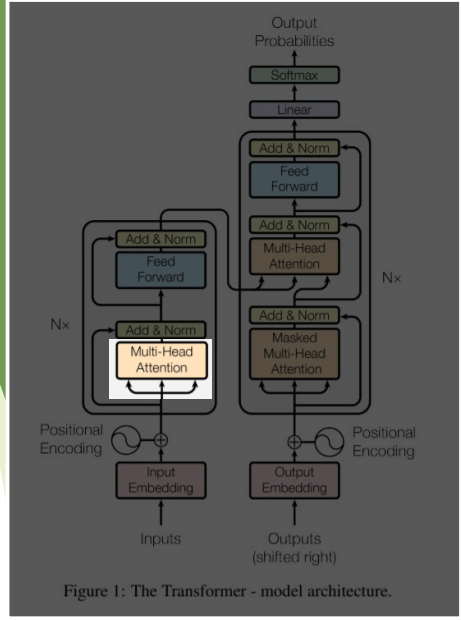


Figure 1: The Transformer - model architecture.

# Multi-Head Self-Attention

## Self-Attention

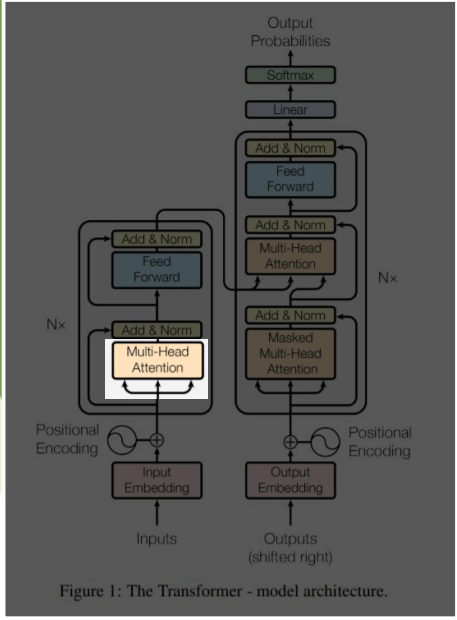


Figure 1: The Transformer - model architecture.

| | | $W^q$ | $W^k$ | $W^v$ |
|---|---|---|---|---|
| $X^p_1$ d=512 | → | $Q_1$ d=64 | $K_1$ d=64 | $V_1$ d=64 |
| $x^p_2$ d=512 | → | $Q_2$ d=64 | $K_2$ d=64 | $V_2$ d=64 |
| $x^p_3$ d=512 | → | $Q_3$ d=64 | $K_3$ d=64 | $V_3$ d=64 |
| $x^p_4$ d=512 | → | $Q_4$ d=64 | $K_4$ d=64 | $V_4$ d=64 |

# Multi-Head Self-Attention

## Self-Attention

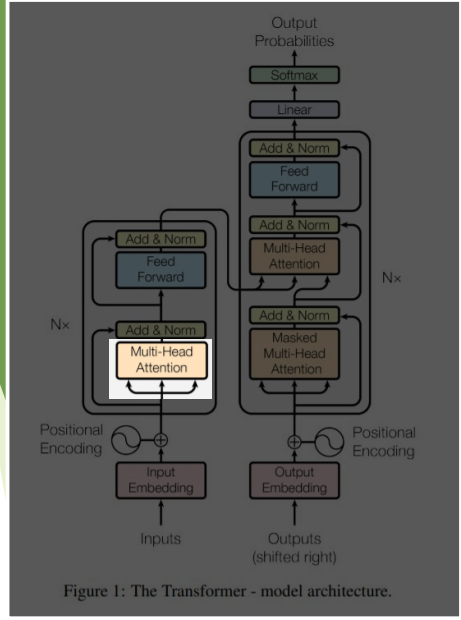
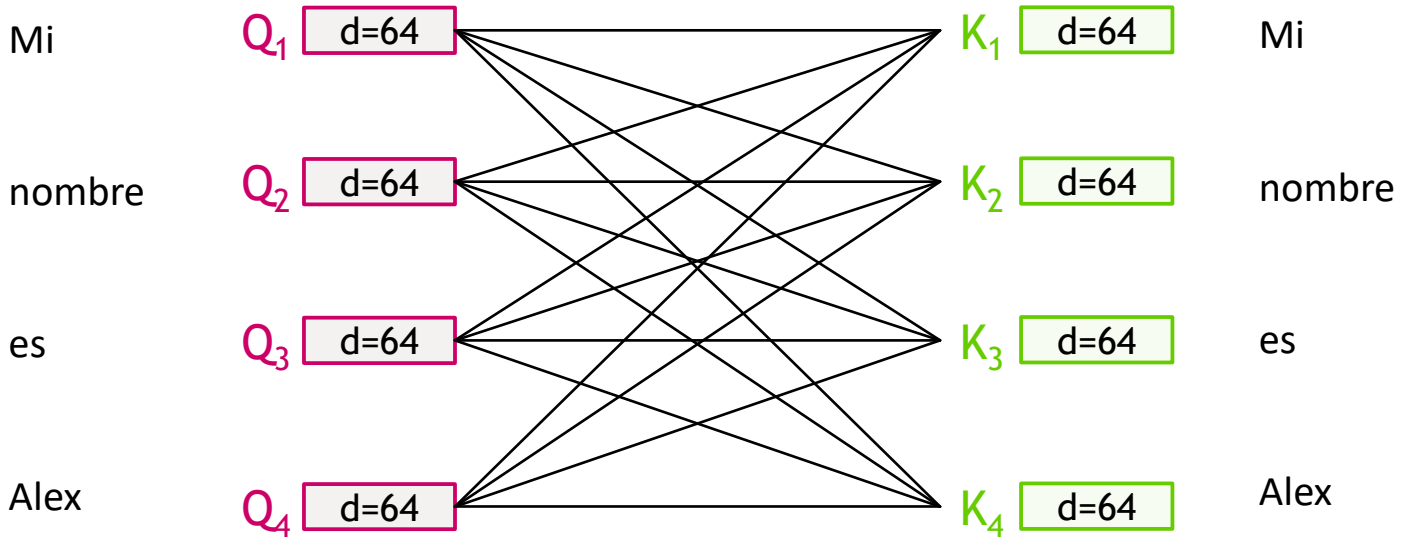


Figure 1: The Transformer - model architecture.



# Multi-Head Self-Attention

## Self-Attention

Example

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

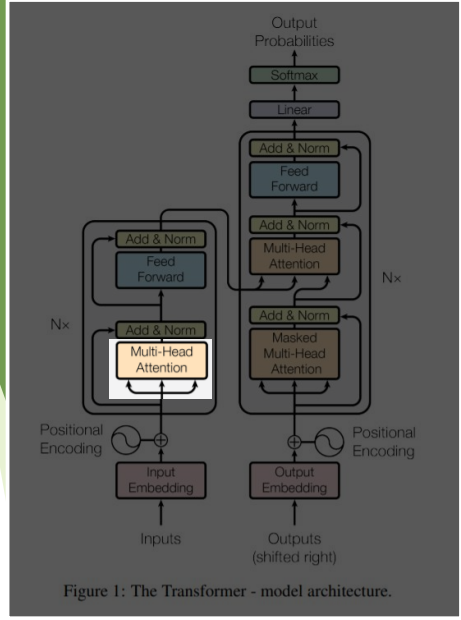


Figure 1: The Transformer - model architecture.

| | | | | | |
|---|---|---|---|---|---|
| Mi | $Q_1$ | d=64 | $K_1$ | d=64 | Mi | Q1*K1 = 50 |
| | | | $K_2$ | d=64 | nombre | Q1*K2 = 79 |
| | | | $K_3$ | d=64 | es | Q1*K3 = 12 |
| | | | $K_4$ | d=64 | Alex | Q1*K4 = 23 |

# Multi-Head Self-Attention

## Self-Attention

Example

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

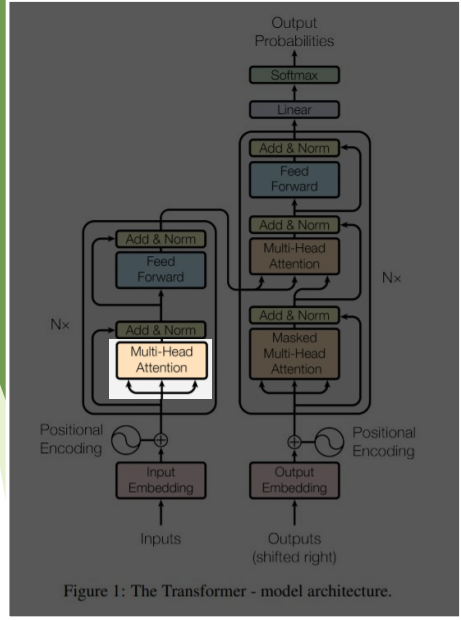


Figure 1: The Transformer - model architecture.

| | | | | | $\frac{1}{\sqrt{d_k}}$ | SoftMax |
|---|---|---|---|---|---|---|
| Mi | $Q_1$ d=64 | $K_1$ d=64 | Mi | Q1*K1 = 50 | 50/8 = 6.25 | 0.0259 |
| | | $K_2$ d=64 | nombre | Q1*K2 = 79 | 79/8 = 9.875 | 0.9730 |
| | | $K_3$ d=64 | es | Q1*K3 = 12 | 12/8 = 1.5 | 0.0002 |
| | | $K_4$ d=64 | Alex | Q1*K4 = 23 | 23/8 = 2.875 | 0.0009 |

# Multi-Head Self-Attention

## Self-Attention

Example

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

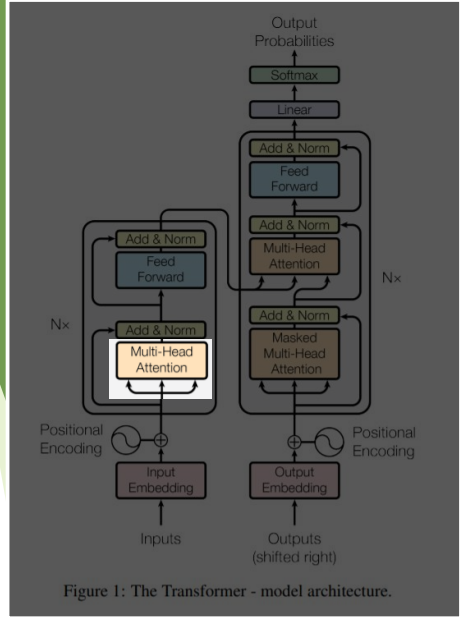


Figure 1: The Transformer - model architecture.

SoftMax

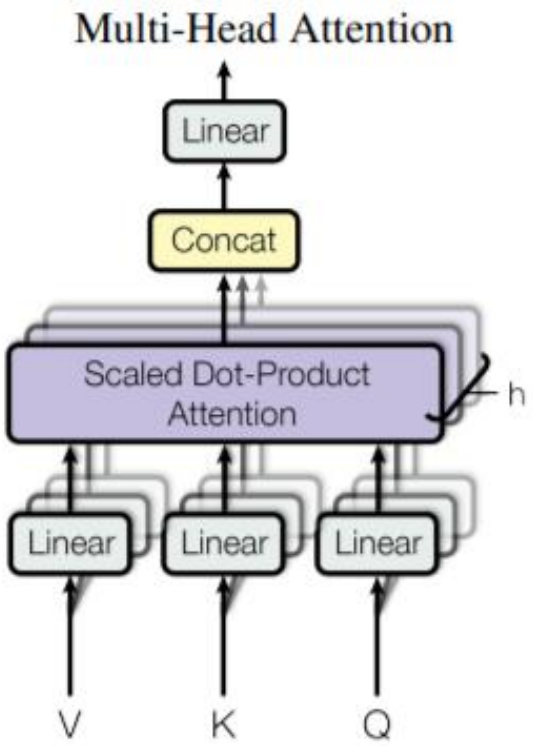
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Mi | $Q_1$ | d=64 | $K_1$ | d=64 | Mi | 0.0259 | * | $V_1$ d=64 |
| | | | $K_2$ | d=64 | nombre | 0.9730 | * | $V_2$ d=64 |
| | | | $K_3$ | d=64 | es | 0.0002 | * | $V_3$ d=64 |
| | | | $K_4$ | d=64 | Alex | 0.0009 | * | $V_4$ d=64 |

$Z^{H0}_1$ d=64

# Multi-Head Self-Attention

## Multi-Head

8 Heads




$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

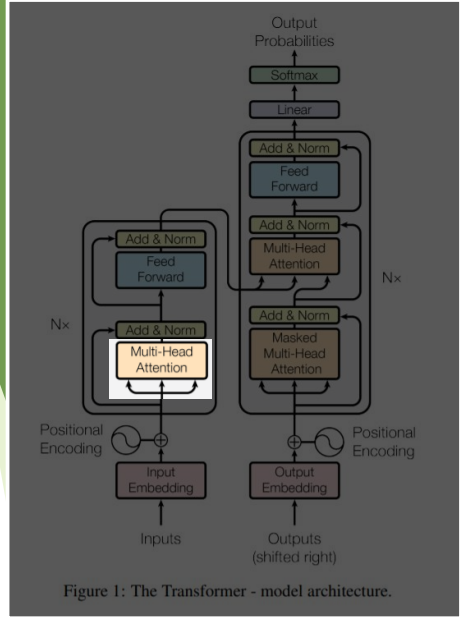


Figure 1: The Transformer - model architecture.

# Multi-Head Self-Attention

## Multi-Head

Example

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

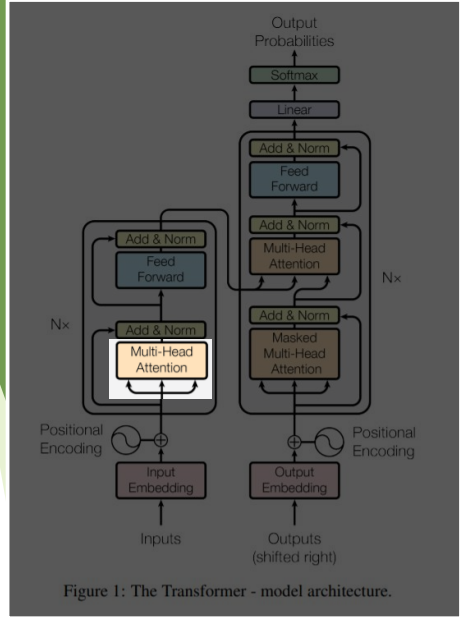


Figure 1: The Transformer - model architecture.

Mi $Z^{H0}_1$ d=64 $Z^{H1}_1$ d=64 $Z^{H2}_1$ d=64 $Z^{H3}_1$ d=64 $Z^{H4}_1$ d=64 $Z^{H5}_1$ d=64 $Z^{H6}_1$ d=64 $Z^{H7}_1$ d=64

nombre $Z^{H0}_1$ d=64 $Z^{H1}_2$ d=64 $Z^{H2}_2$ d=64 $Z^{H3}_2$ d=64 $Z^{H4}_2$ d=64 $Z^{H5}_2$ d=64 $Z^{H6}_2$ d=64 $Z^{H7}_2$ d=64

es $Z^{H0}_3$ d=64 $Z^{H1}_3$ d=64 $Z^{H2}_3$ d=64 $Z^{H3}_3$ d=64 $Z^{H4}_3$ d=64 $Z^{H5}_3$ d=64 $Z^{H6}_3$ d=64 $Z^{H7}_3$ d=64

Alex $Z^{H0}_3$ d=64 $Z^{H1}_3$ d=64 $Z^{H2}_3$ d=64 $Z^{H3}_3$ d=64 $Z^{H4}_3$ d=64 $Z^{H5}_3$ d=64 $Z^{H6}_3$ d=64 $Z^{H7}_3$ d=64

* $W^o$ =

# Multi-Head Self-Attention

## Multi-Head

Example

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

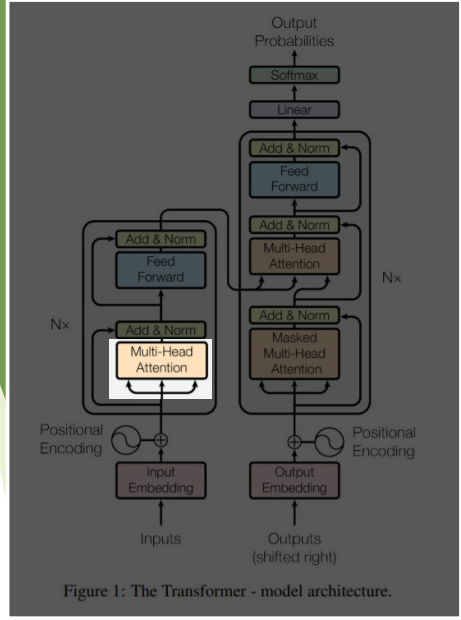


Figure 1: The Transformer - model architecture.

| | | |
|---|---|---|
| Mi | $Z_1$ | d=512 |
| nombre | $Z_2$ | d=512 |
| es | $Z_3$ | d=512 |
| Alex | $Z_4$ | d=512 |

# Add & Normal

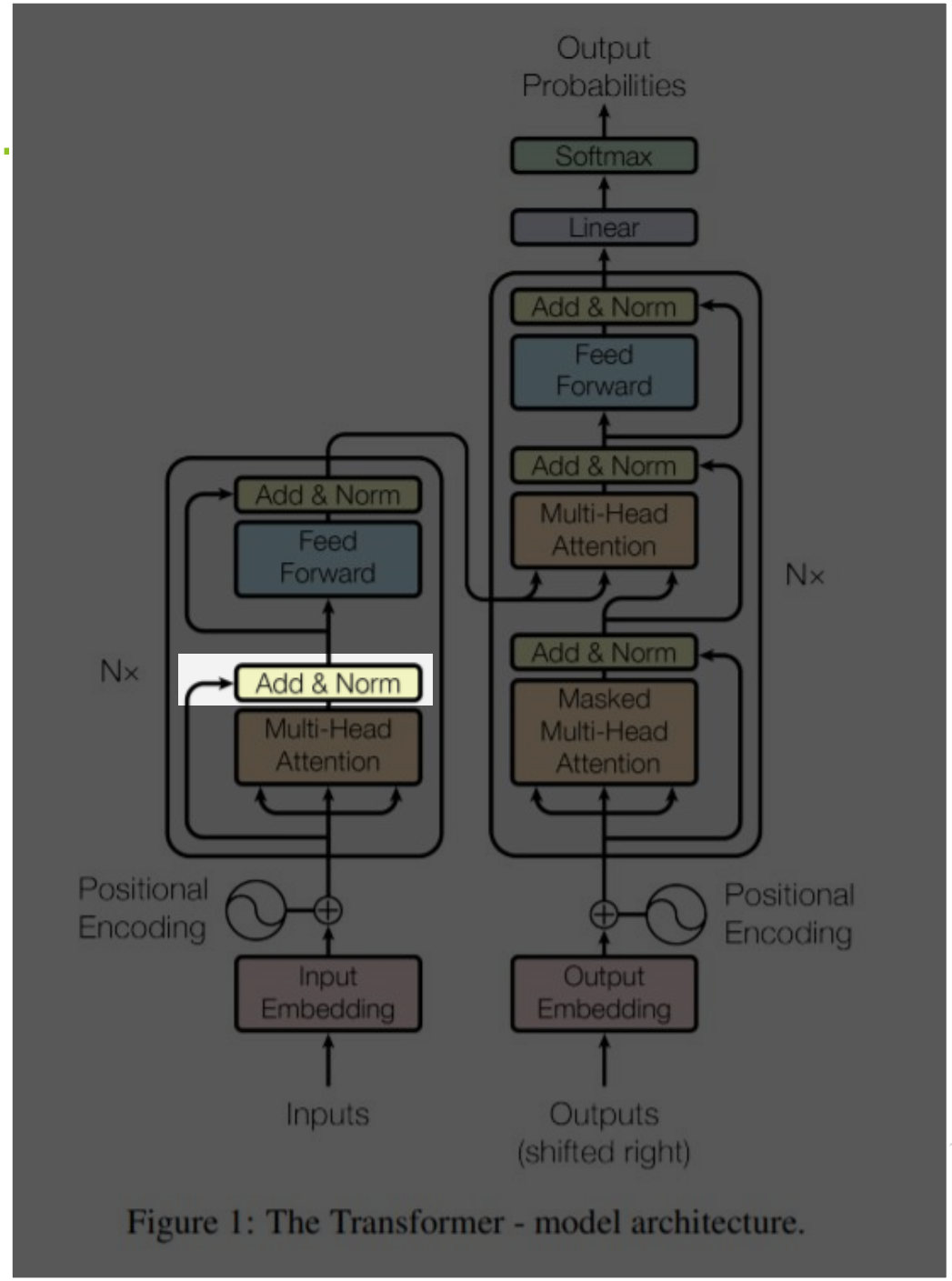



Figure 1: The Transformer - model architecture.

# Add & Normalize

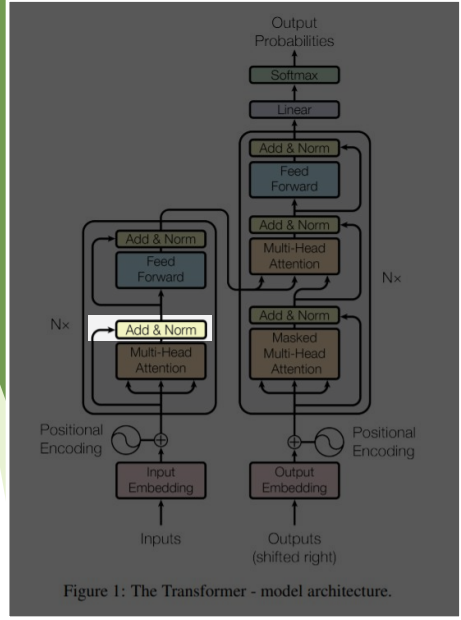


Figure 1: The Transformer - model architecture.

Mi

nombre

es

Alex

Norm ( $X^p_1$ d=512 + $Z_1$ d=512 ) = $Z^n_1$ d=512

( $x^p_2$ d=512 + $Z_2$ d=512 ) = $Z^n_2$ d=512

( $x^p_3$ d=512 + $Z_3$ d=512 ) = $Z^n_3$ d=512

( $x^p_4$ d=512 + $Z_4$ d=512 ) = $Z^n_4$ d=512

# Feed Forward

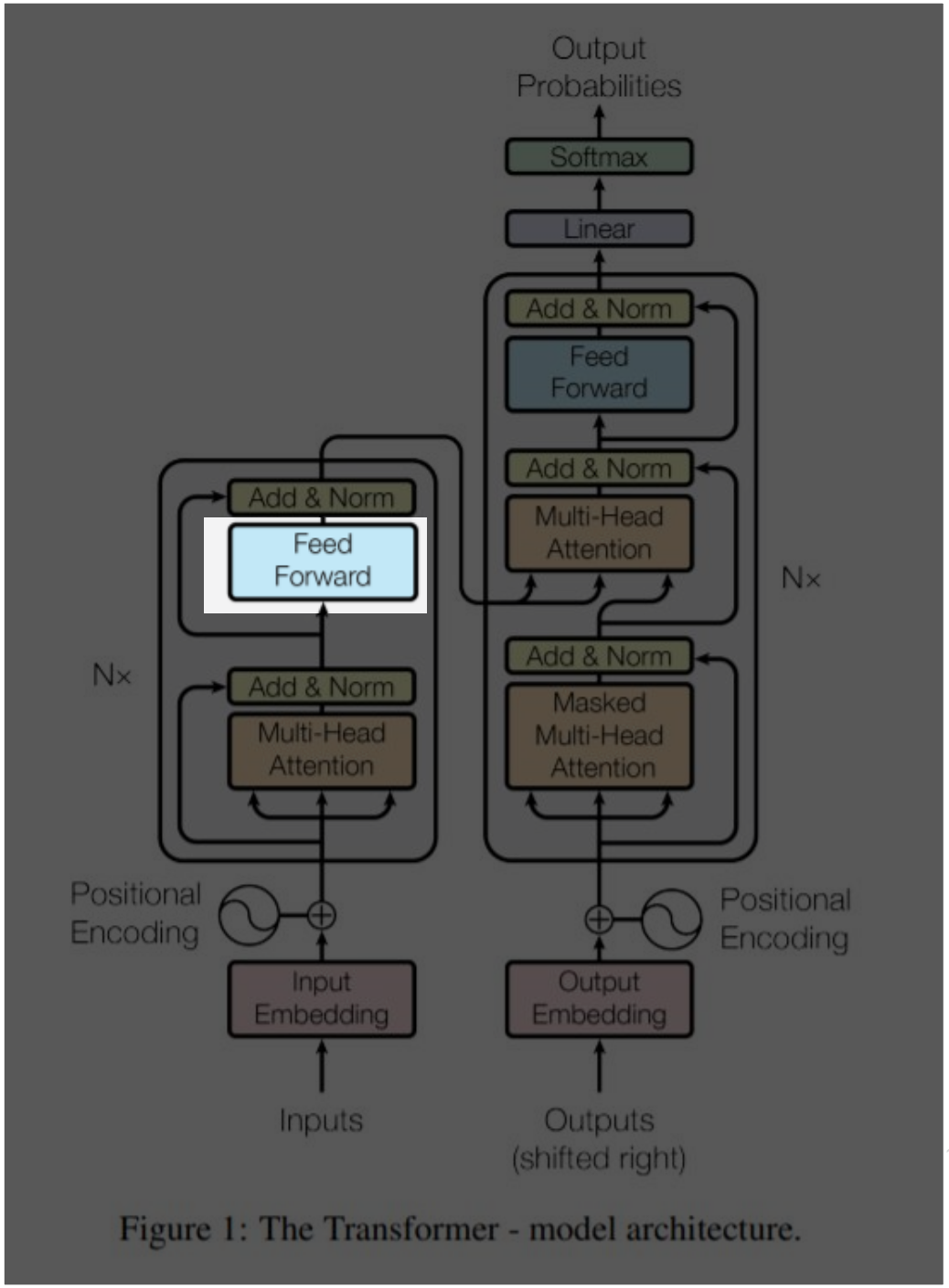
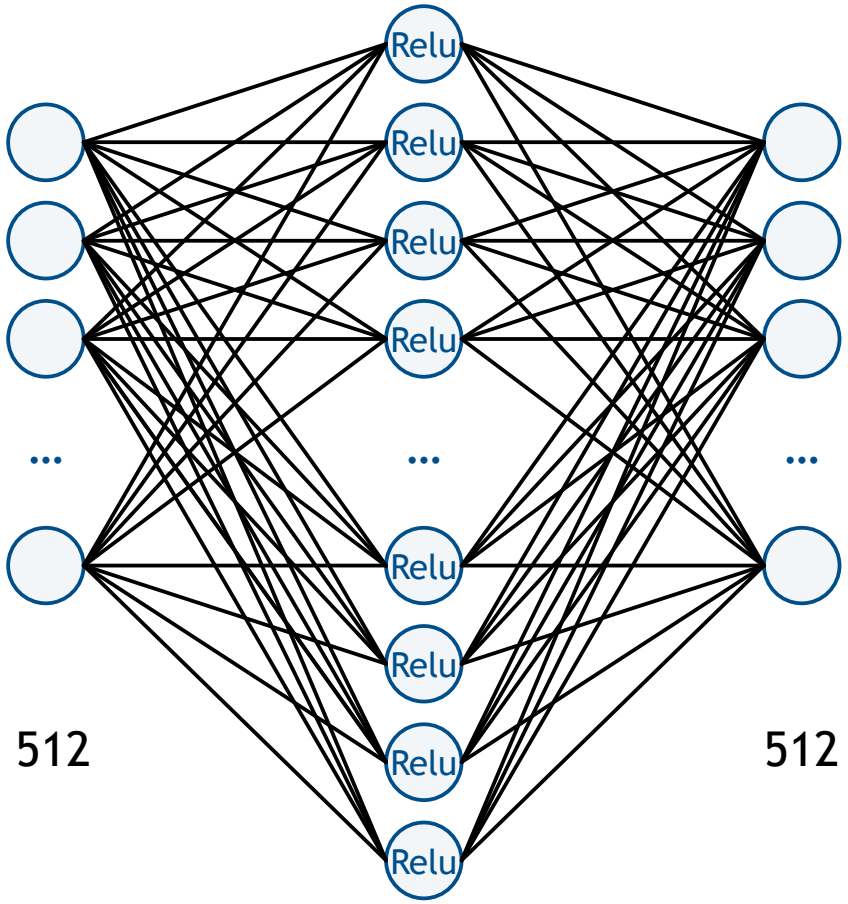



Figure 1: The Transformer - model architecture.

# Feed Forward Neural Network

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$




The FFNN is applied to each word separately

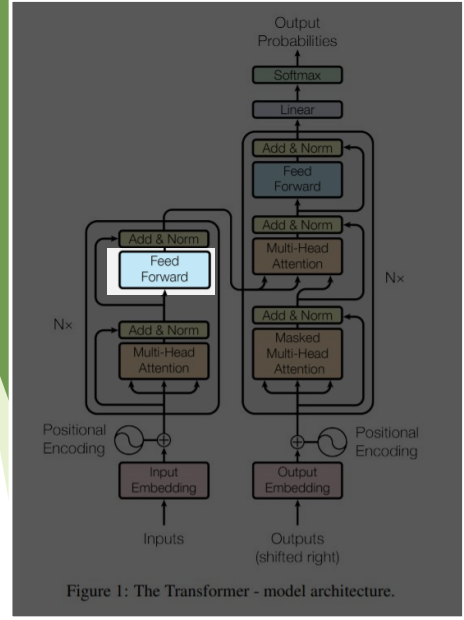
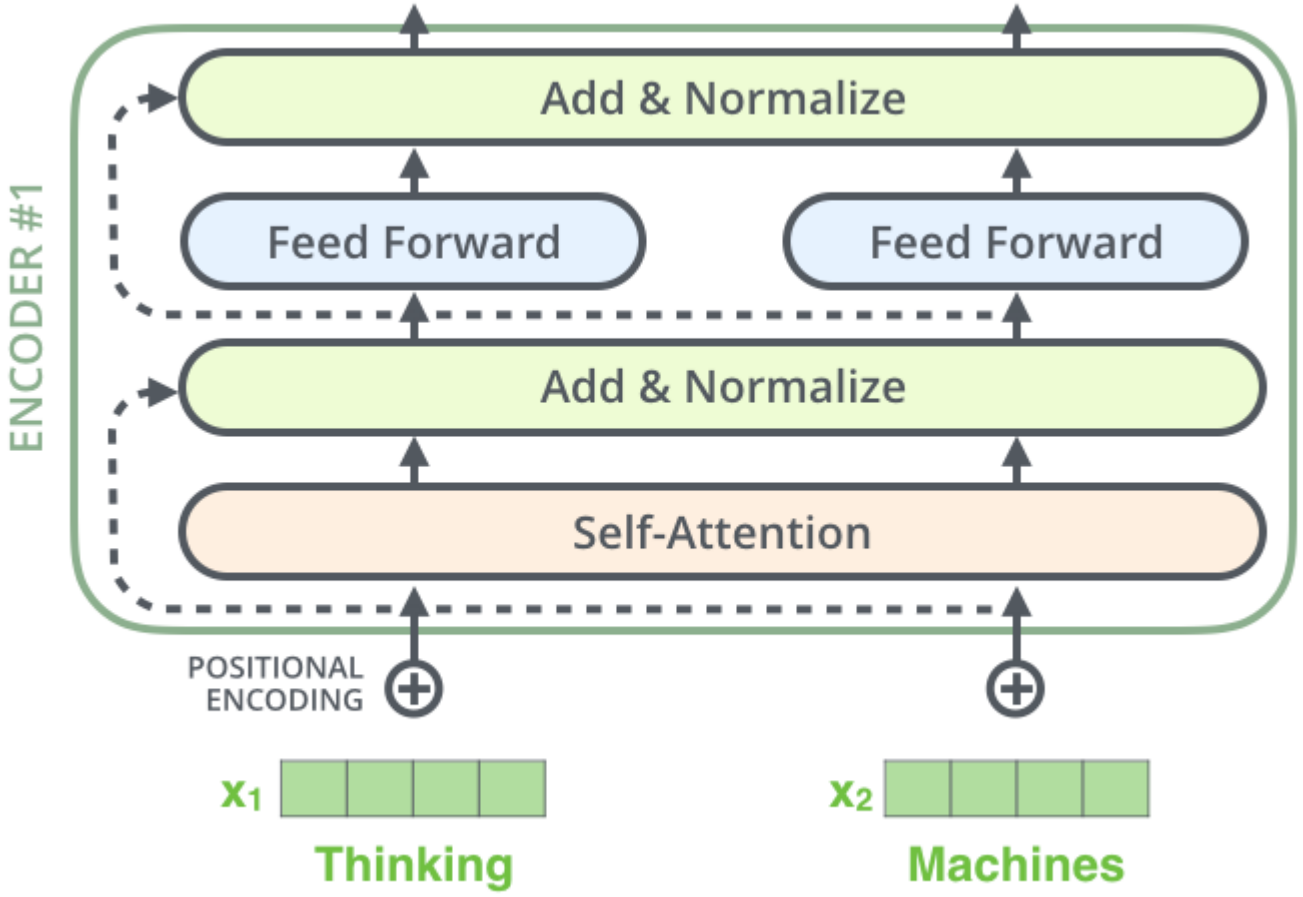


Figure 1: The Transformer - model architecture.

# Recap




Source: Jay Alammar

# Encoder-Deco

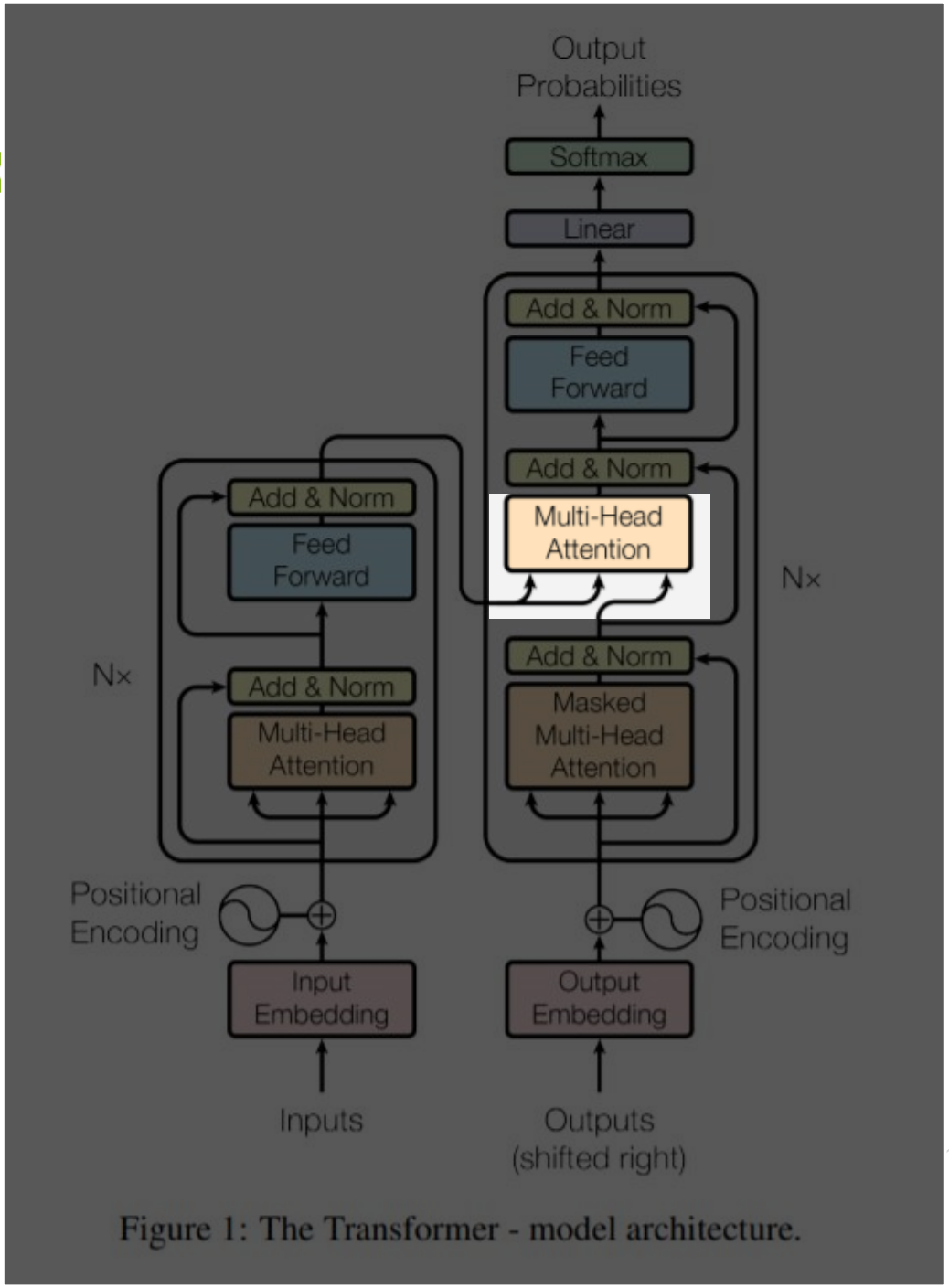
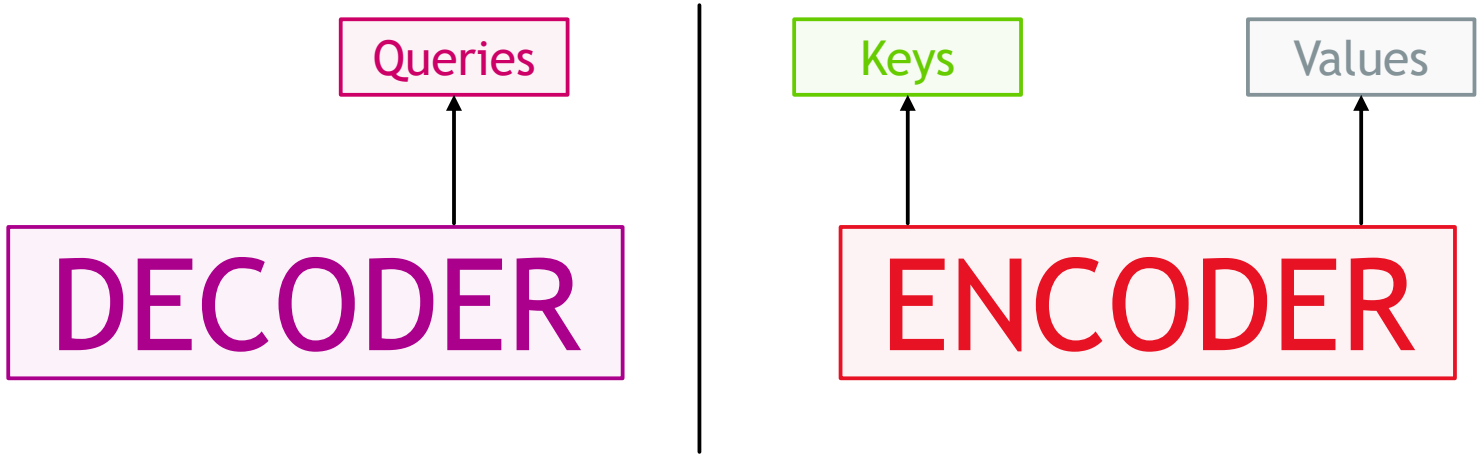



Figure 1: The Transformer - model architecture.

# Encoder-Decoder M-H S-A

The sublayers needs 3 vectors for each word (d=64)



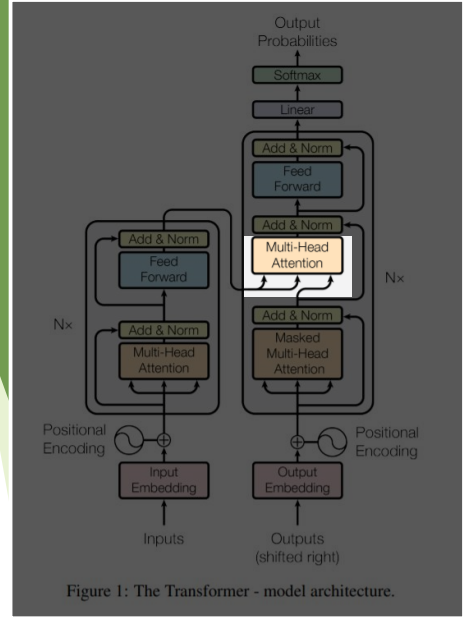
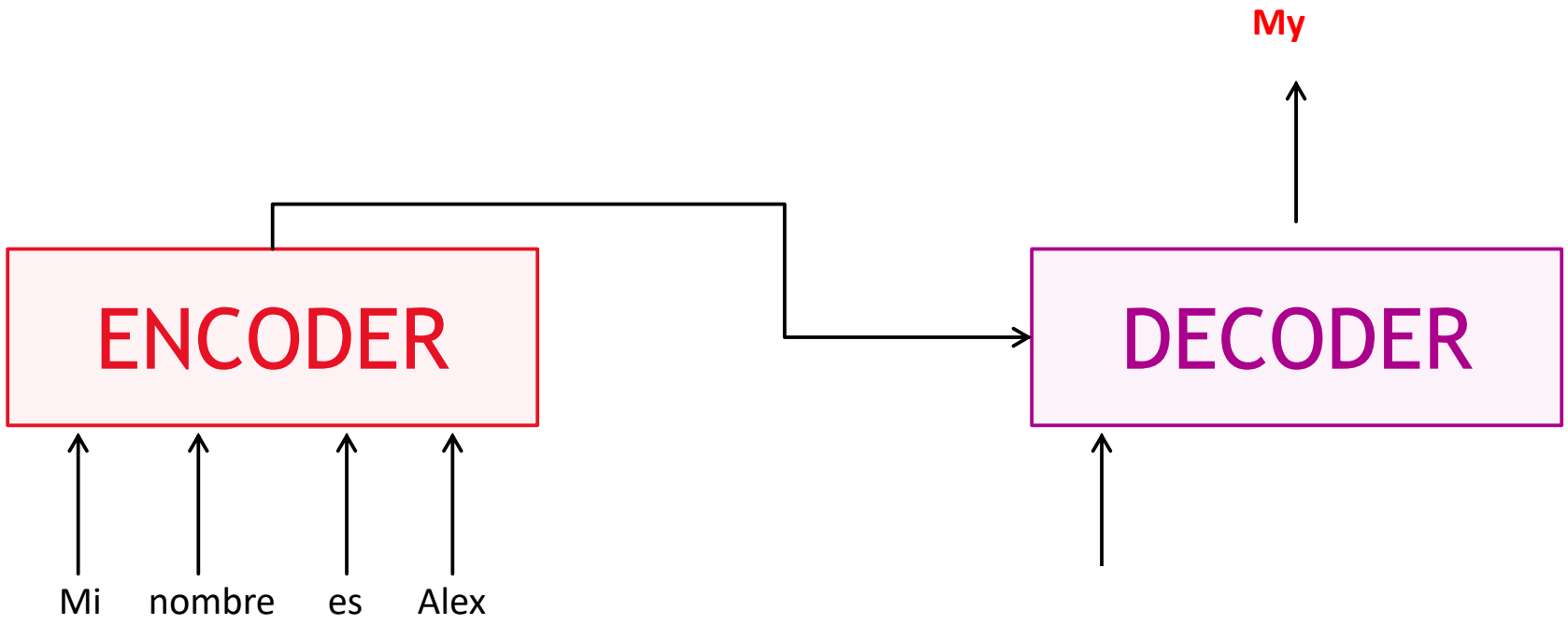
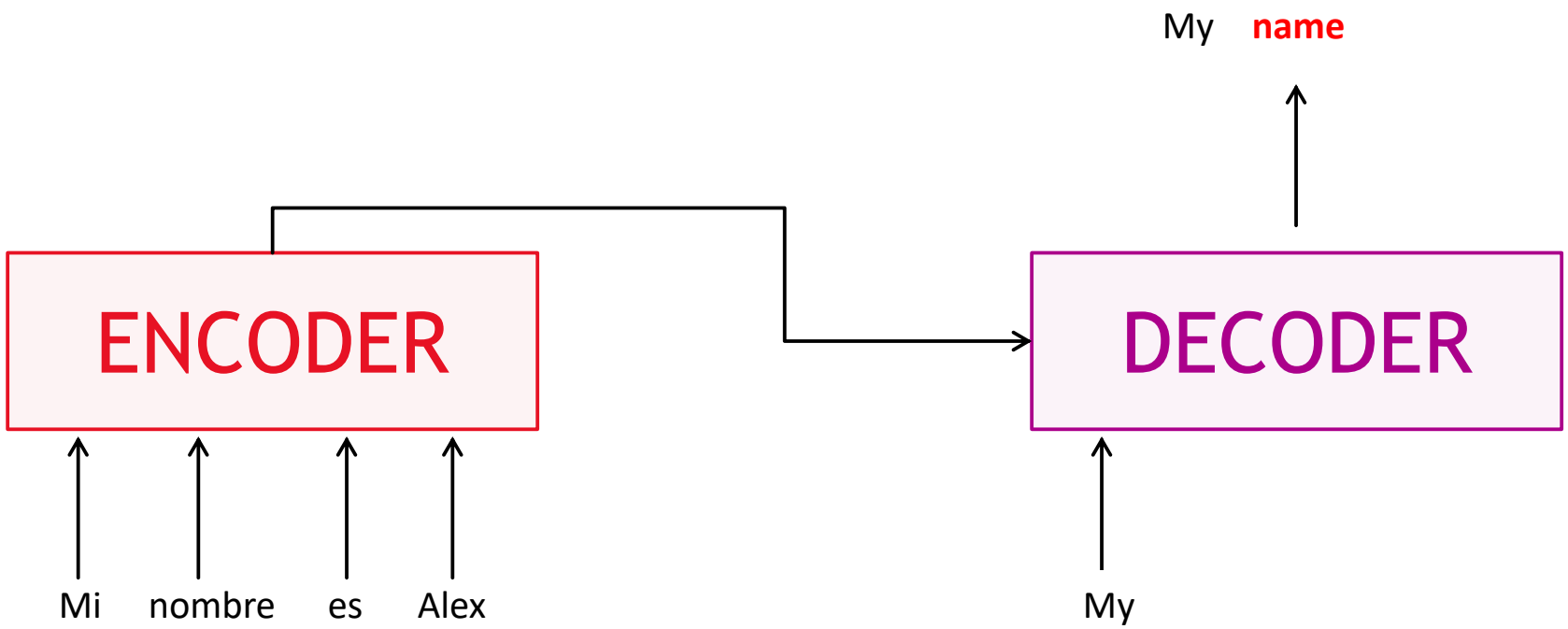



Figure 1: The Transformer - model architecture.

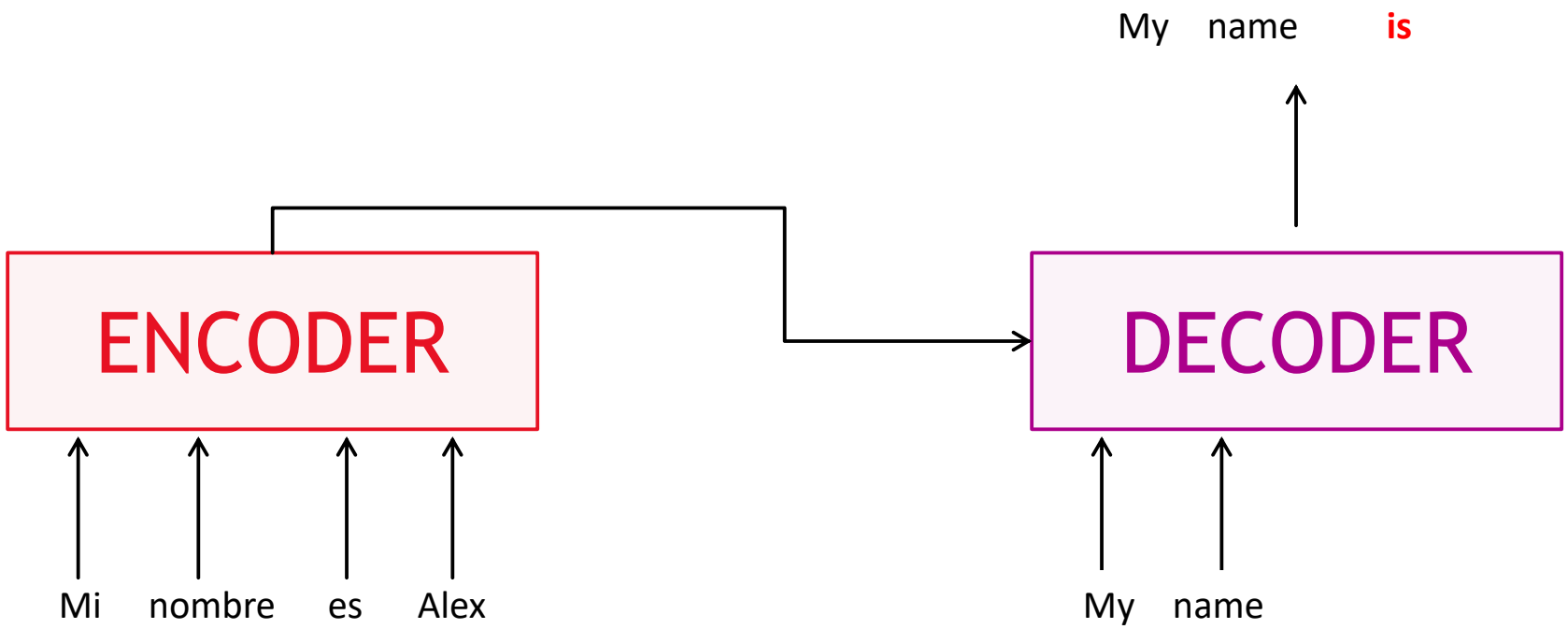
# EXAMPLE



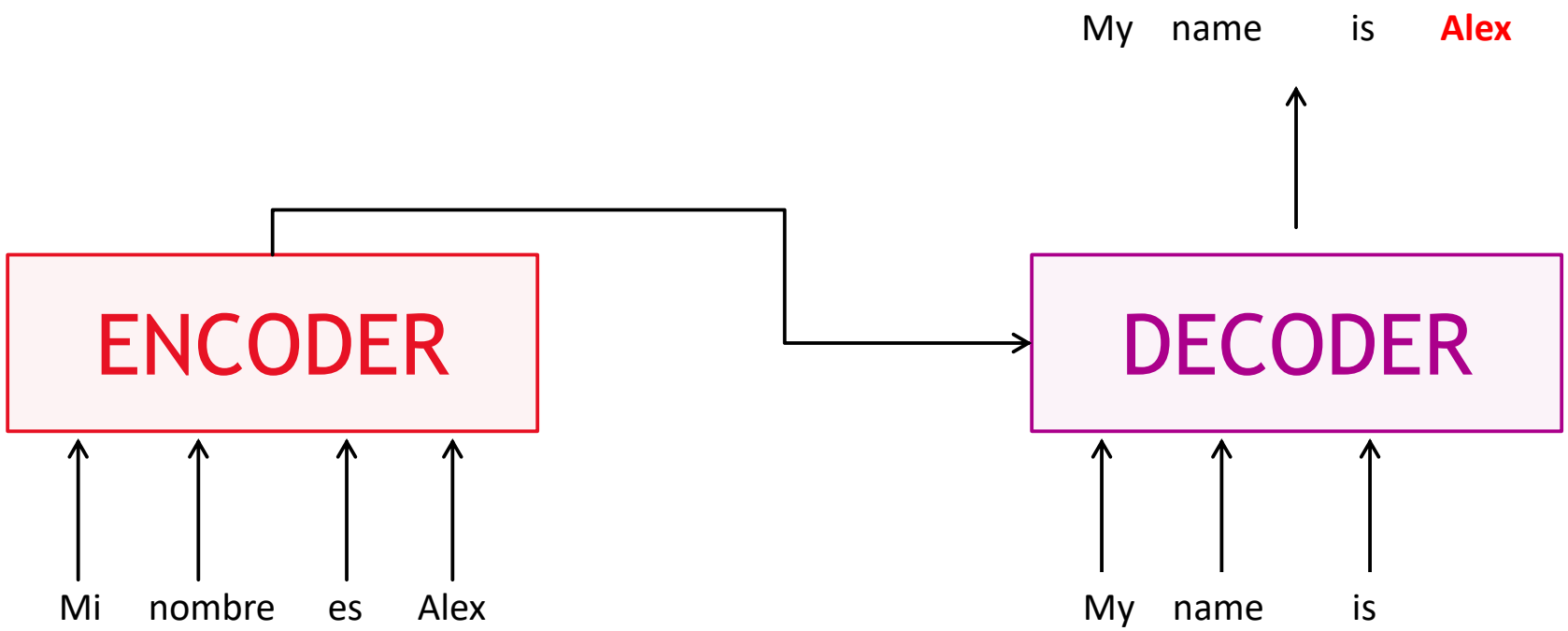
# EXAMPLE



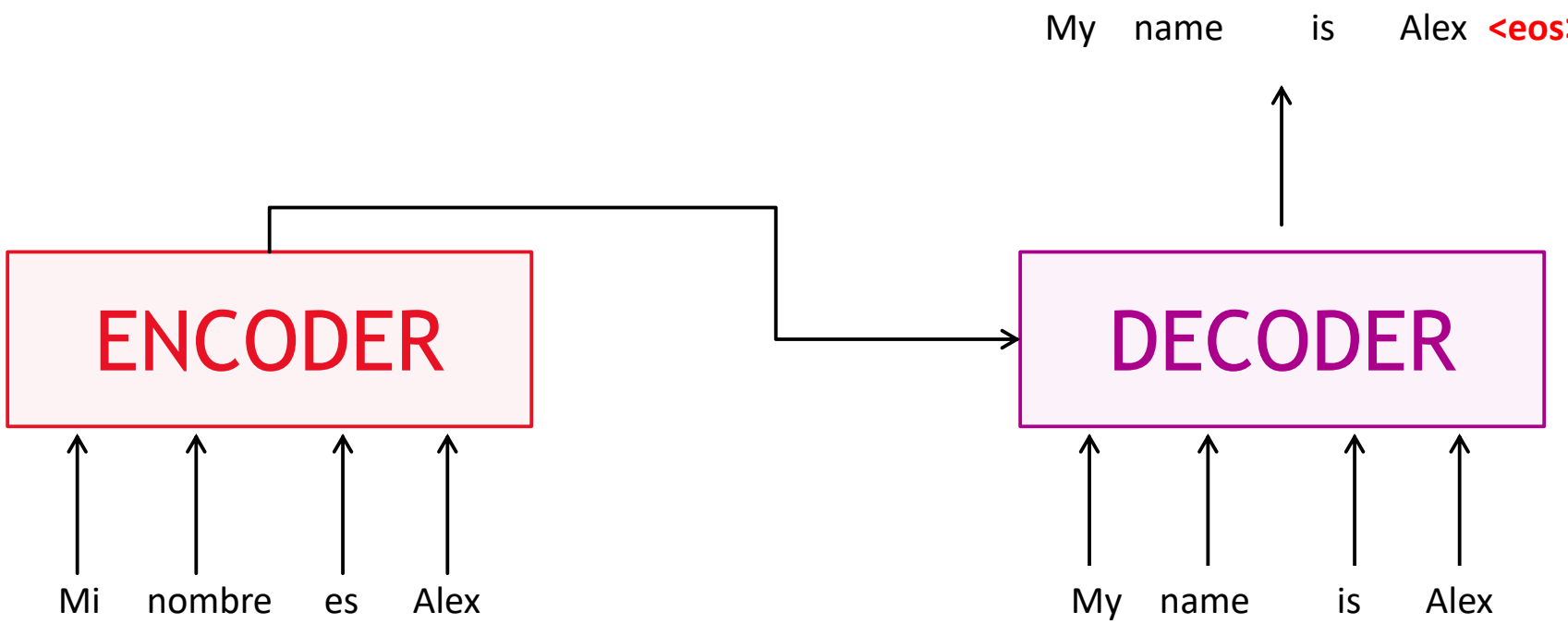
# EXAMPLE



# EXAMPLE



# EXAMPLE


My name is Alex <eos>
ENCODER
DECODER
Mi nombre es Alex
My name is Alex

# Optimizer

We used the Adam optimizer [17] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (3)$$

This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

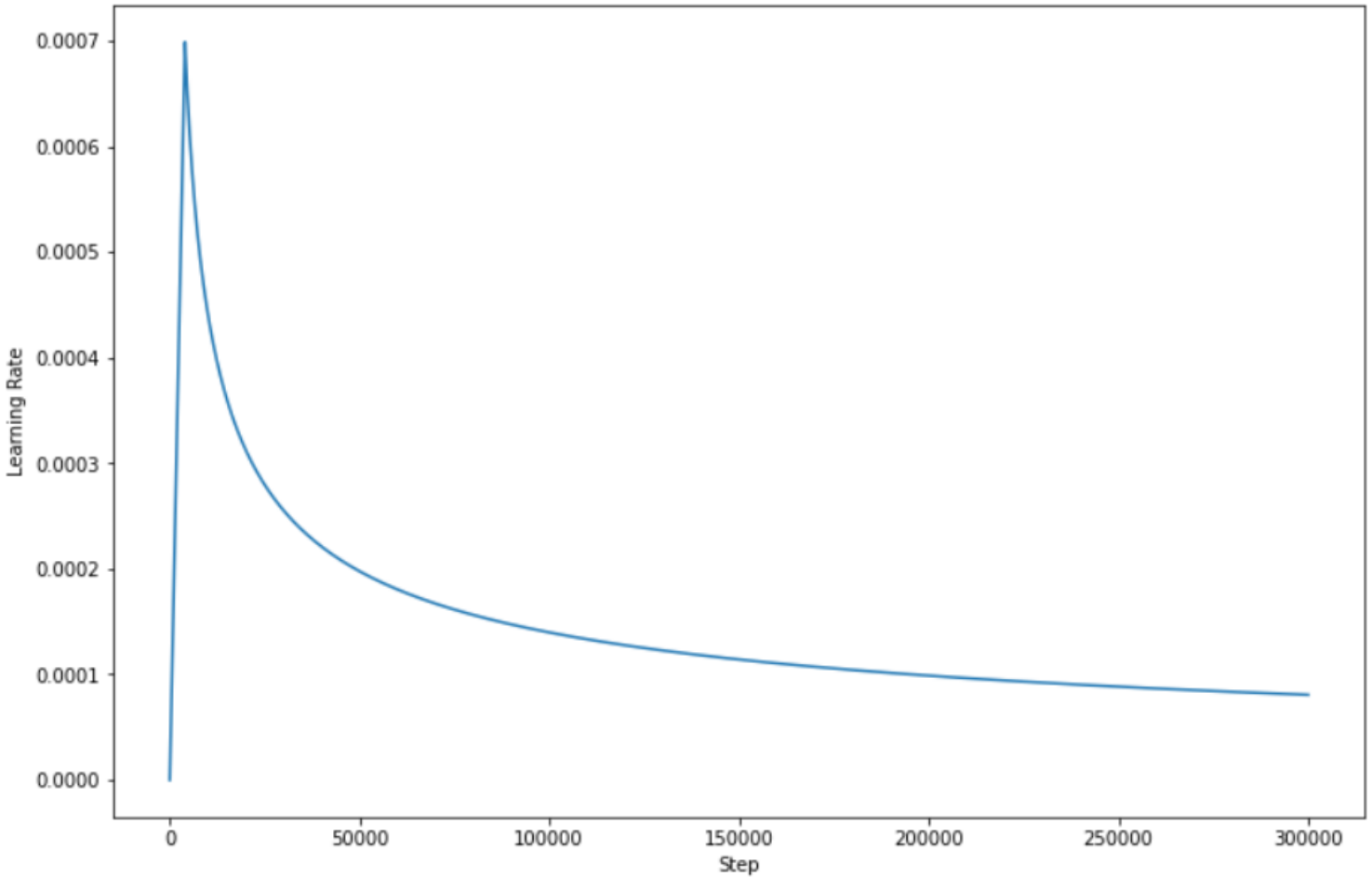

0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
Learning Rate
0
50000
100000
150000
200000
250000
300000
Step

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [15] | 23.75 | | | |
| Deep-Att + PosUnk [32] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [31] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [8] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [26] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [32] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [31] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [8] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | **$3.3 \cdot 10^{18}$** | |
| Transformer (big) | **28.4** | **41.0** | $2.3 \cdot 10^{19}$ | |

| | $N$ | $d_{\text{model}}$ | $d_{\text{ff}}$ | $h$ | $d_k$ | $d_v$ | $P_{drop}$ | $\epsilon_{ls}$ | train steps | PPL (dev) | BLEU (dev) | params $\times 10^6$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| base | 6 | 512 | 2048 | 8 | 64 | 64 | 0.1 | 0.1 | 100K | 4.92 | 25.8 | 65 |
| (A) | | | | 1 | 512 | 512 | | | | 5.29 | 24.9 | |
| | | | | 4 | 128 | 128 | | | | 5.00 | 25.5 | |
| | | | | 16 | 32 | 32 | | | | 4.91 | 25.8 | |
| | | | | 32 | 16 | 16 | | | | 5.01 | 25.4 | |
| (B) | | | | | 16 | | | | | 5.16 | 25.1 | 58 |
| | | | | | 32 | | | | | 5.01 | 25.4 | 60 |
| (C) | 2 | | | | | | | | | 6.11 | 23.7 | 36 |
| | 4 | | | | | | | | | 5.19 | 25.3 | 50 |
| | 8 | | | | | | | | | 4.88 | 25.5 | 80 |
| | | 256 | | | 32 | 32 | | | | 5.75 | 24.5 | 28 |
| | | 1024 | | | 128 | 128 | | | | 4.66 | 26.0 | 168 |
| | | | 1024 | | | | | | | 5.12 | 25.4 | 53 |
| | | | 4096 | | | | | | | 4.75 | 26.2 | 90 |
| (D) | | | | | | | 0.0 | | | 5.77 | 24.6 | |
| | | | | | | | 0.2 | | | 4.95 | 25.5 | |
| | | | | | | | | 0.0 | | 4.67 | 25.3 | |
| | | | | | | | | 0.2 | | 5.47 | 25.7 | |
| (E) | positional embedding instead of sinusoids | | | | | | | | | 4.92 | 25.7 | |
| big | 6 | 1024 | 4096 | 16 | | | 0.3 | | 300K | **4.33** | **26.4** | 213 |

In Table 3 rows (B), we observe that reducing the attention key size $d_k$ hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [8], and observe nearly identical results to the base model.

# END

- https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- https://jalammar.github.io/illustrated-transformer/
- https://colab.research.google.com/github/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb#scrollTo=SB9-56JWKv6T
- https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC