Transformers: "Attention is all you need"

Table of Contents

Alejandro C Parra Garcia

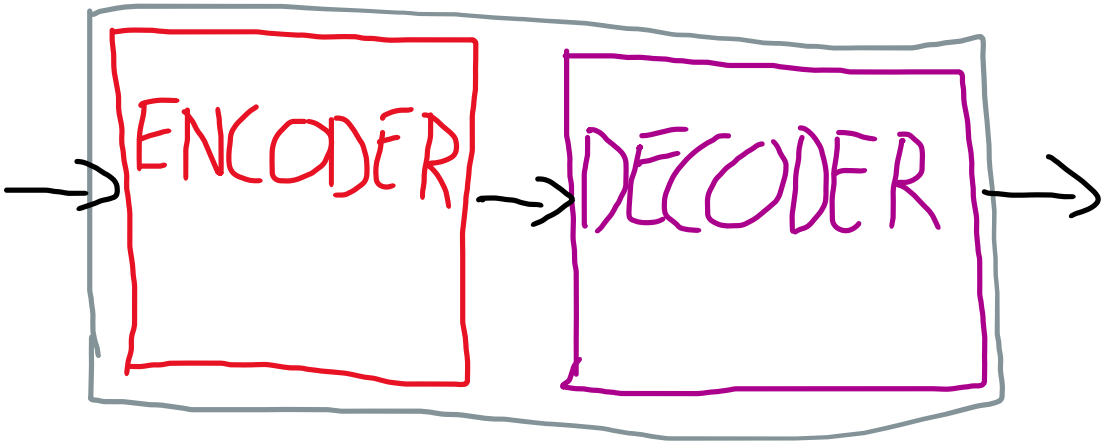
Translation model

Input in one language → Transformer → Output in another language

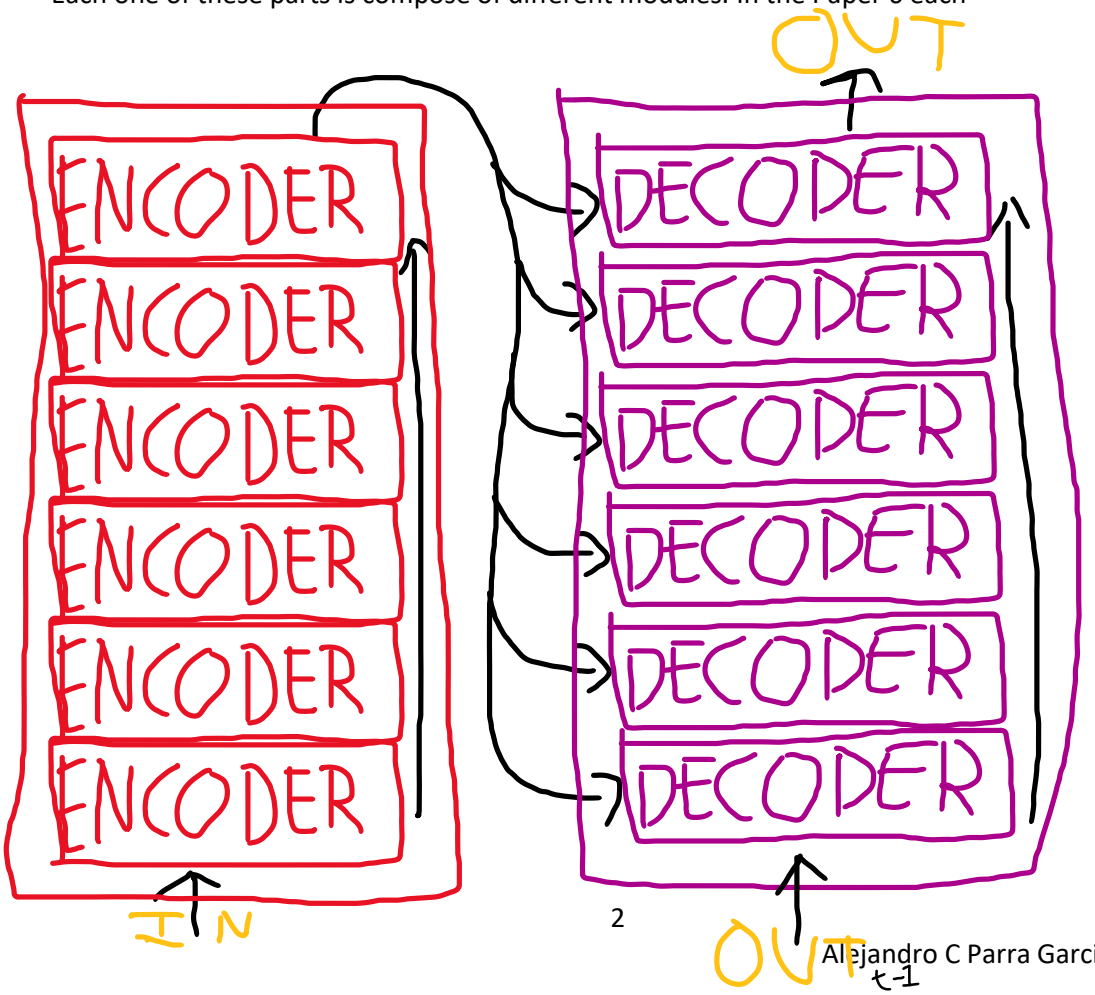
1. Basic Structure

How does the transformer work?

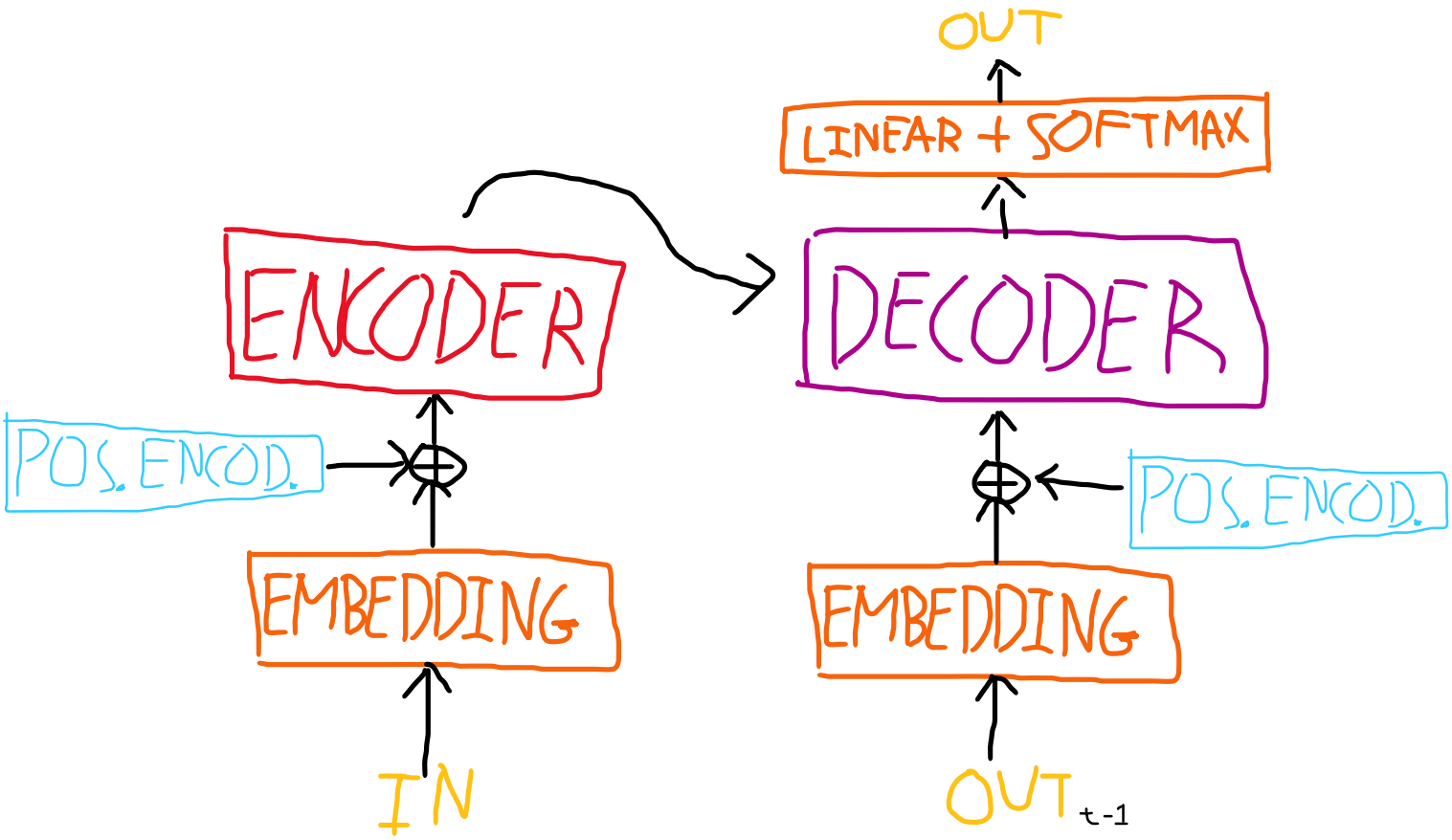
2 parts, encoder and decoder



Each one of these parts is compose of different modules. In the Paper 6 each



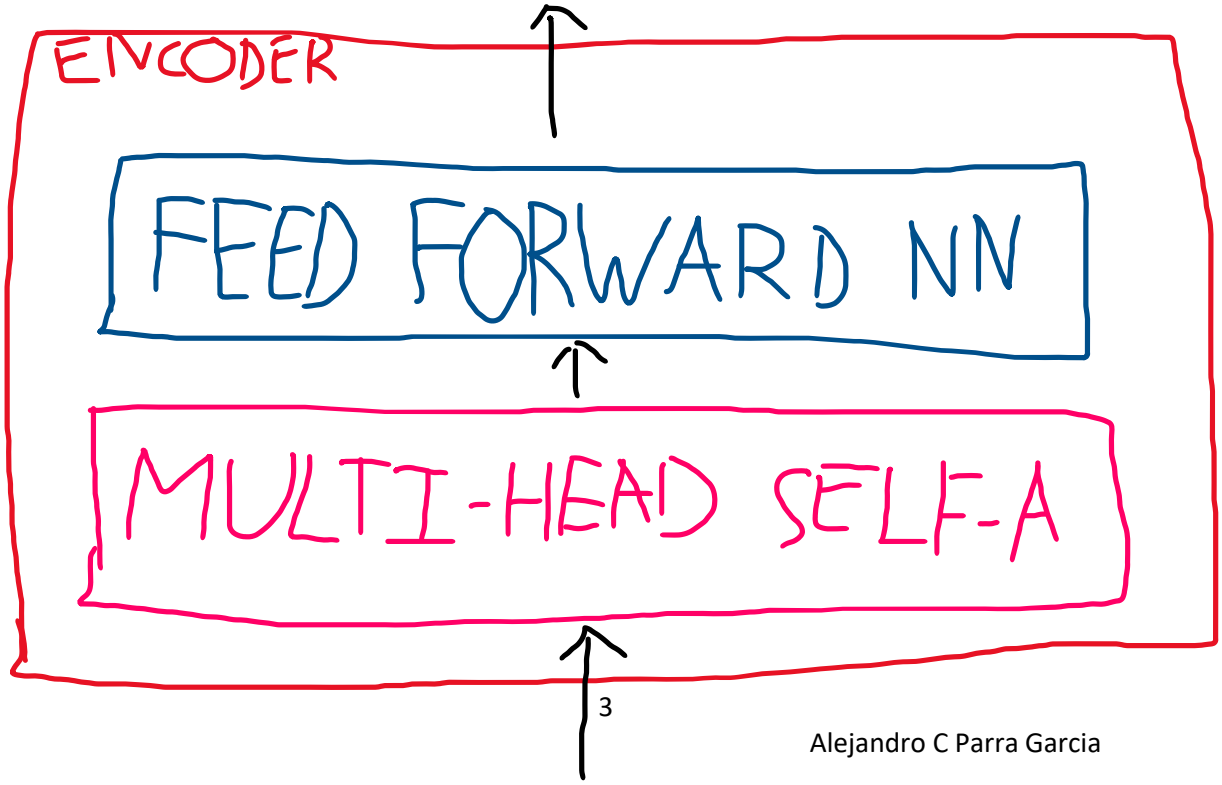
Before the Encoder and Decoder, we need to convert the words to vectors, we use embedding. The output is converted to words using a linear model and a SoftMax function.



The Encoder and Decoder are composed of different sublayers

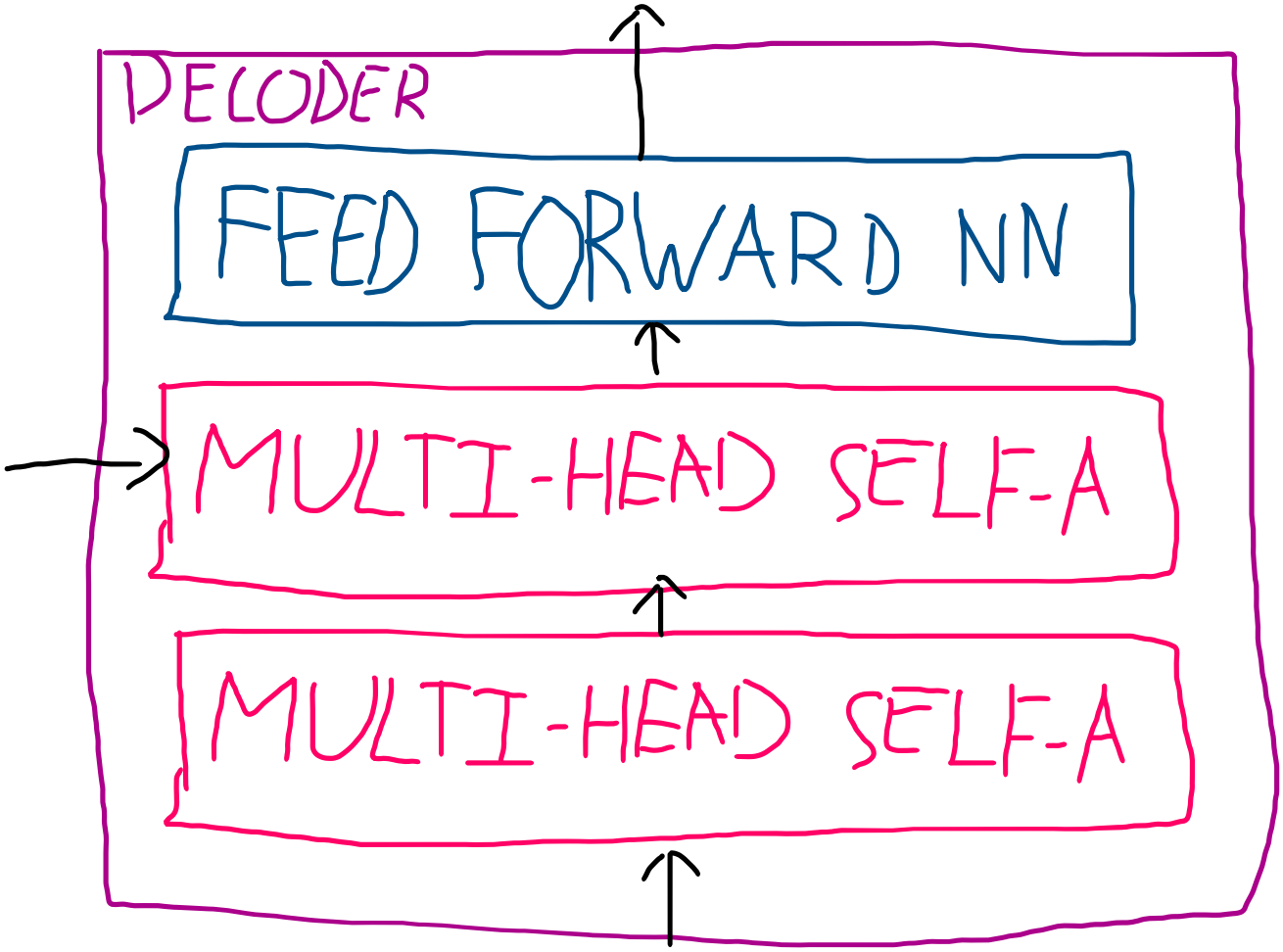
1.1. Encoder

The **Encoder** has 2 sublayers: Multi-head Self-Attention and Feed Forward Neural Network



1.2. Decoder

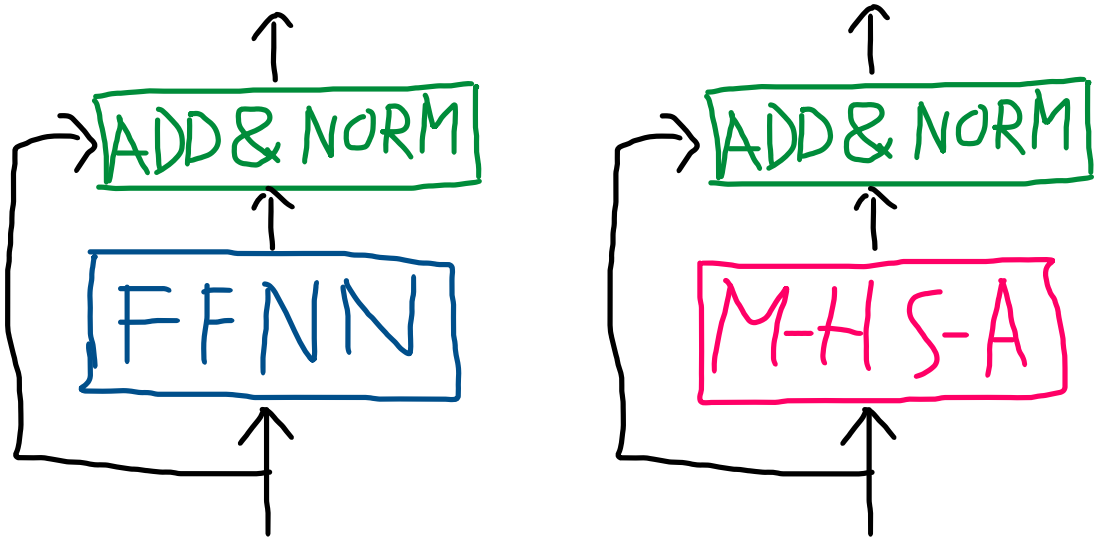
The **Decoder** has 3 sublayers: 2 Multi-head Self-Attention and Feed Forward Neural Network



The Encoder and Decoder work with numbers, vectors, and matrixes. But the input is a phrase of words, we need to convert them into numbers.

1.3. Residuals

Each sublayer have a residual connection around the layer, follow by a layer normalization: LayerNorm(x + Sublayer(x))



2. Complete Structure

Figure from the original Paper: “Attention is all you need”

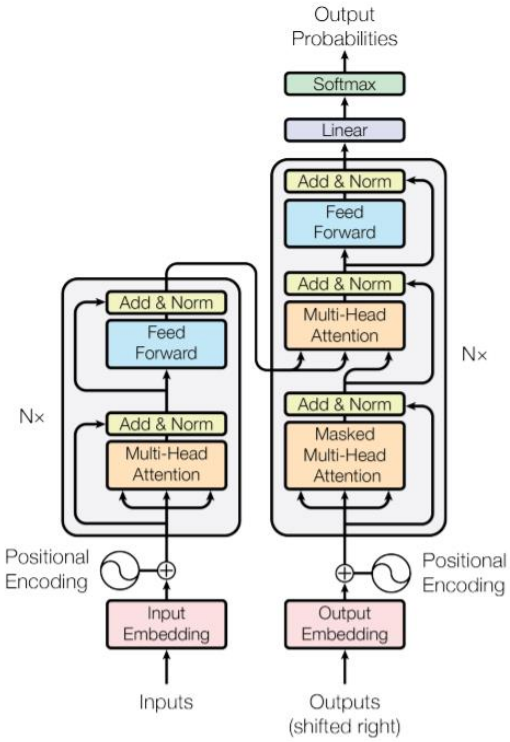


Figure 1: The Transformer - model architecture.

3. How it works

We want to translate the phrase: “Mi nombre es Alex” to English: “My name is Alex”

3.1. Embedding

First, we convert each word into a vector of numbers. (In the paper the dimension is 512, this is also the dimension for the output for each sublayer)

Mi Nombre es Alex

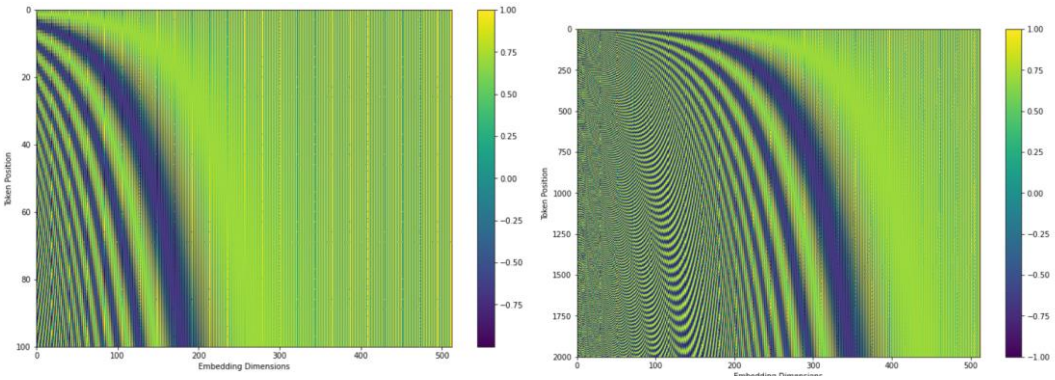
3.2. Positional Encoding

A vector is created representing the position of each word in the phrase (Dimension also 512). The vector is created using these equations:

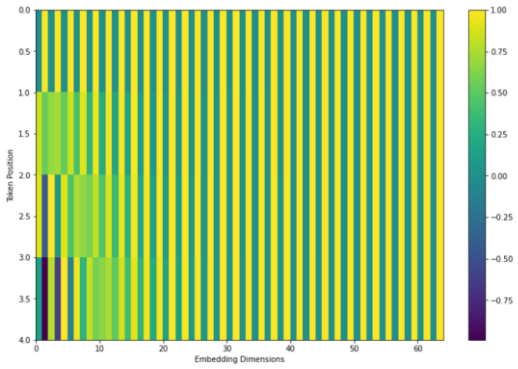
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

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 2π 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}.

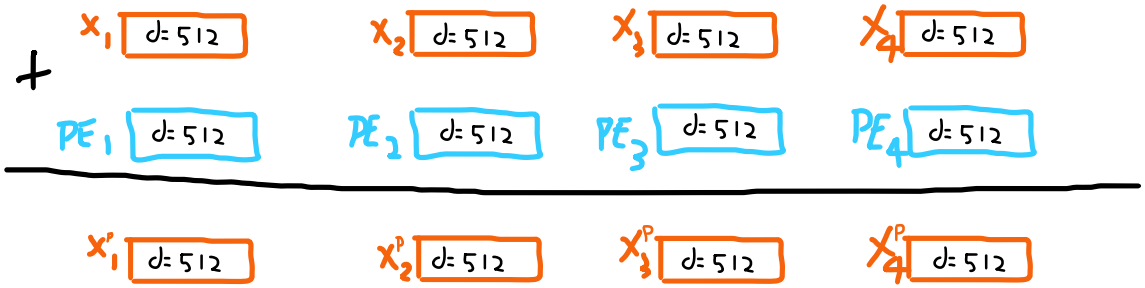
Visualization: by Jay Alammar (github)



Each position is going to have a different vector.



Mi Nombre es Alex

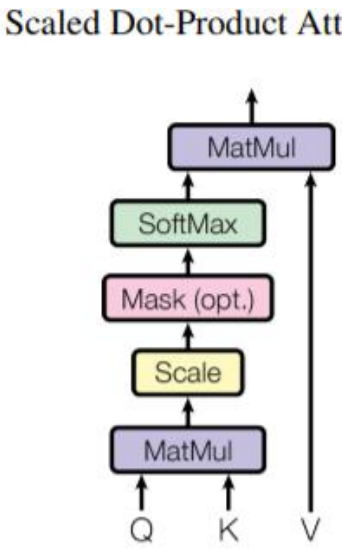


3.3. Multi-Head Self-Attention

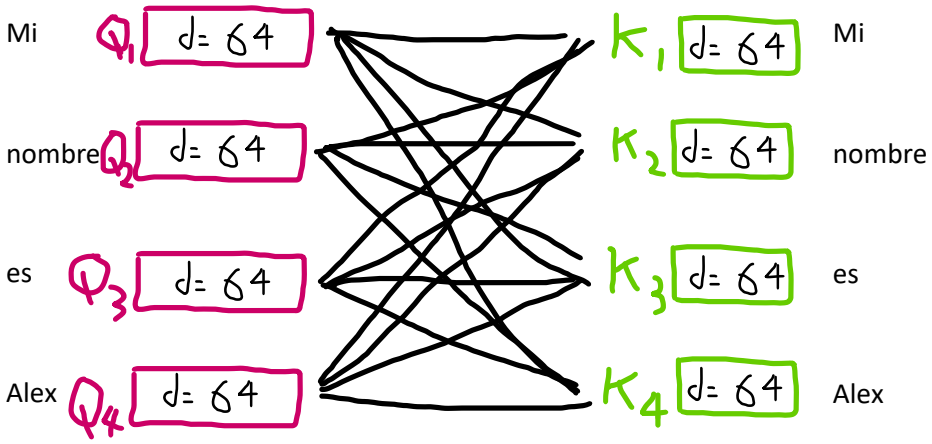
3.3.1. Self-Attention (Scaled Dot-Product Attention)

The objective of the self-attention sublayer is to find the relations between the different words of the phrase no matter what's the distance between the words. Each word is computed to the reast.

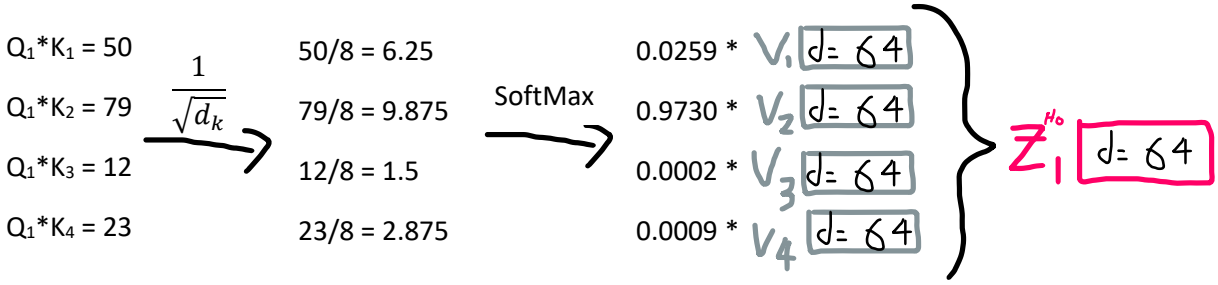
These prevents the "loosing memory" problem of the RNN models. This is the process according to the paper. Q, K, V are vectors of dimension 64.



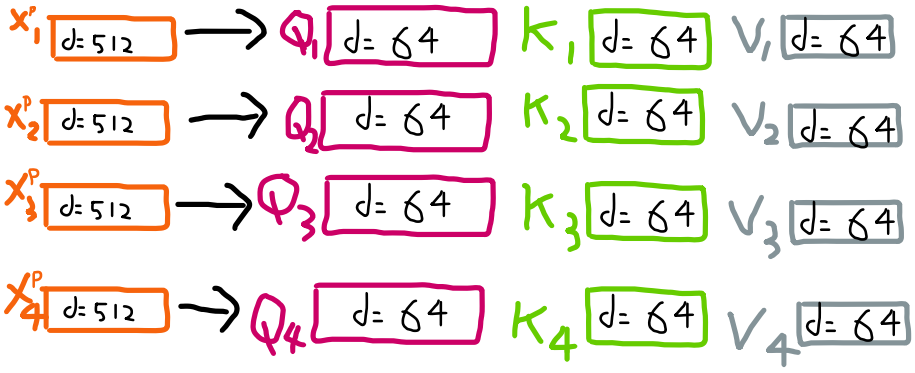
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



For first word ('Mi') we get the next 4 values (Invented numbers). (Repeat for each word)



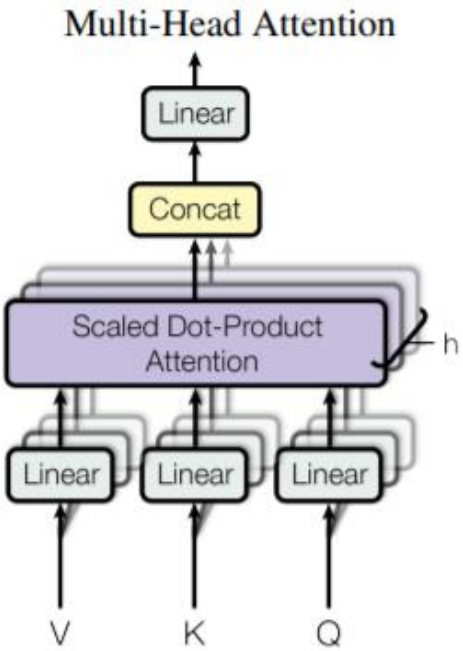
Vectors Q, K, V are obtained by multiplying the input of the sublayer by some matrixes (theses matrixes are parameters the model needs to learn)



3.3.2. Multi-Head

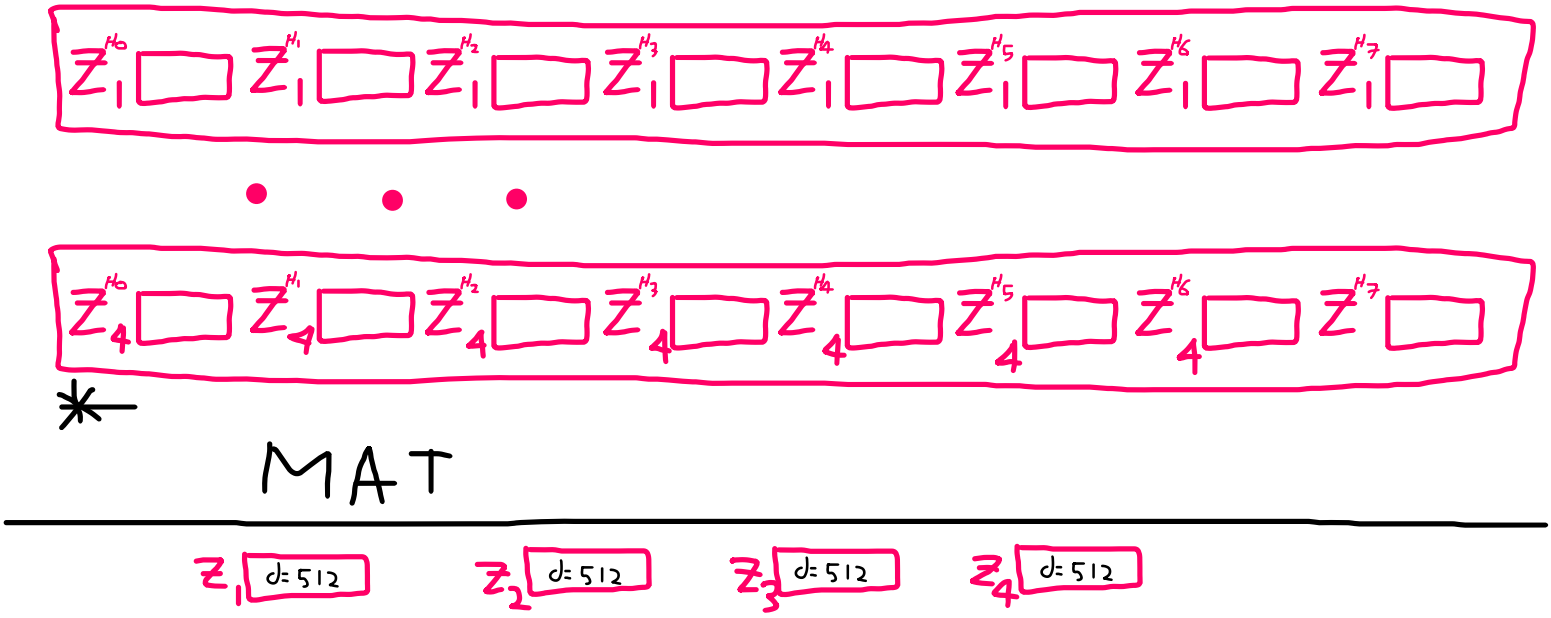
The actual process is done 8 different times with different Q, K, V vectors for the words for each head.

Diagram from the paper:



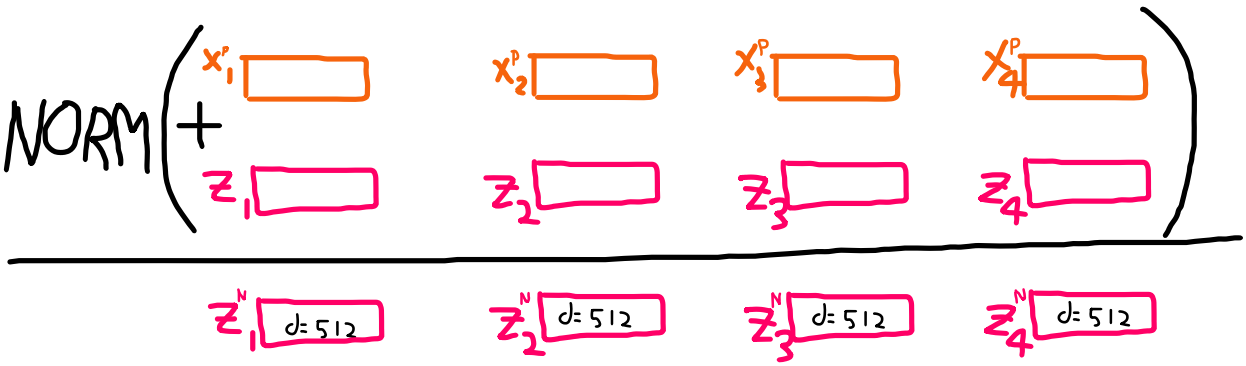
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

First, we need to concatenate all the outputs from each self-attention. We repeat this process for each word. After that we multiply the resulting concatenated vectors by a matrix, and we get the final result



Check this visually using Tensor Flow (Colab)

3.4. Add & Normalize



3.5. Feed Forward Neural Network

Normal Neural Network. 512 Input nodes, 512 Output Nodes, 1 Hidden layer (nodes 2048), ReLu activation function.

The result is a series of vectors, one for each word:

Once it's done, we do again the Add and Normalize, we get this:

3.6. End of Encoder

Once all of the encoders have finished, be get one vector for each word.

Like in the Self-Attention sublayer, we convert these vectors into the K, V vectors. We will use these new vectors for the Multi-Head Self-Attention sublayer in the decoder.

3.7. Decoder

The sublayers in the decoder work the same way as the sublayers in the encoder.

The only difference is that the K, V vectors for the middle sublayer in the decoder comes from the output from the encoder instead than from the output of the previous sublayer.

3.8. Linear & SoftMax

Once the Decoder has finished, we need to convert the output to a word.

First, we used a linear transformation neural network to change the vector length to the length of the vocab. After that a SoftMax is used to create the probabilities.

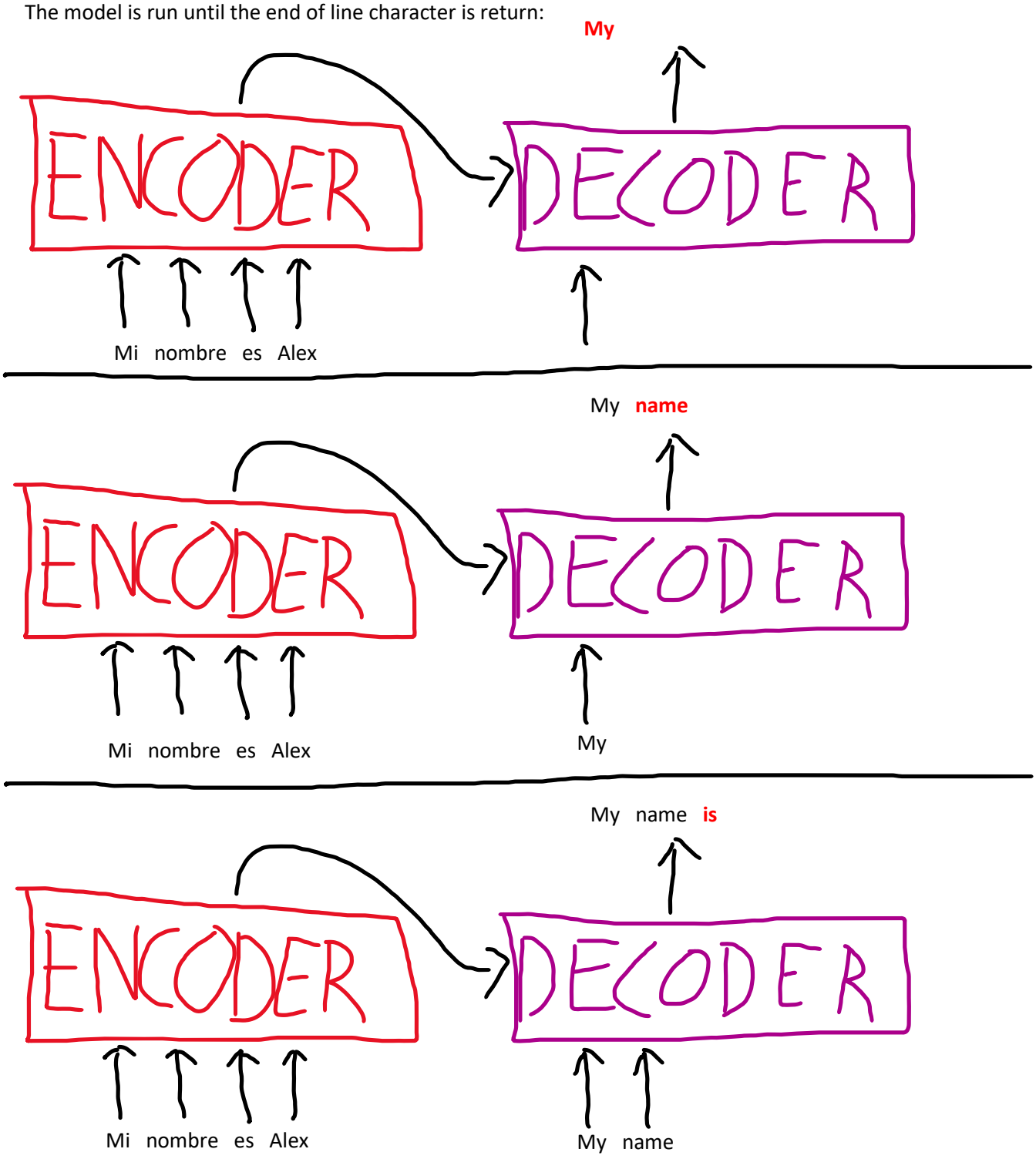
3.9. Greedy Decoding & Beam Search

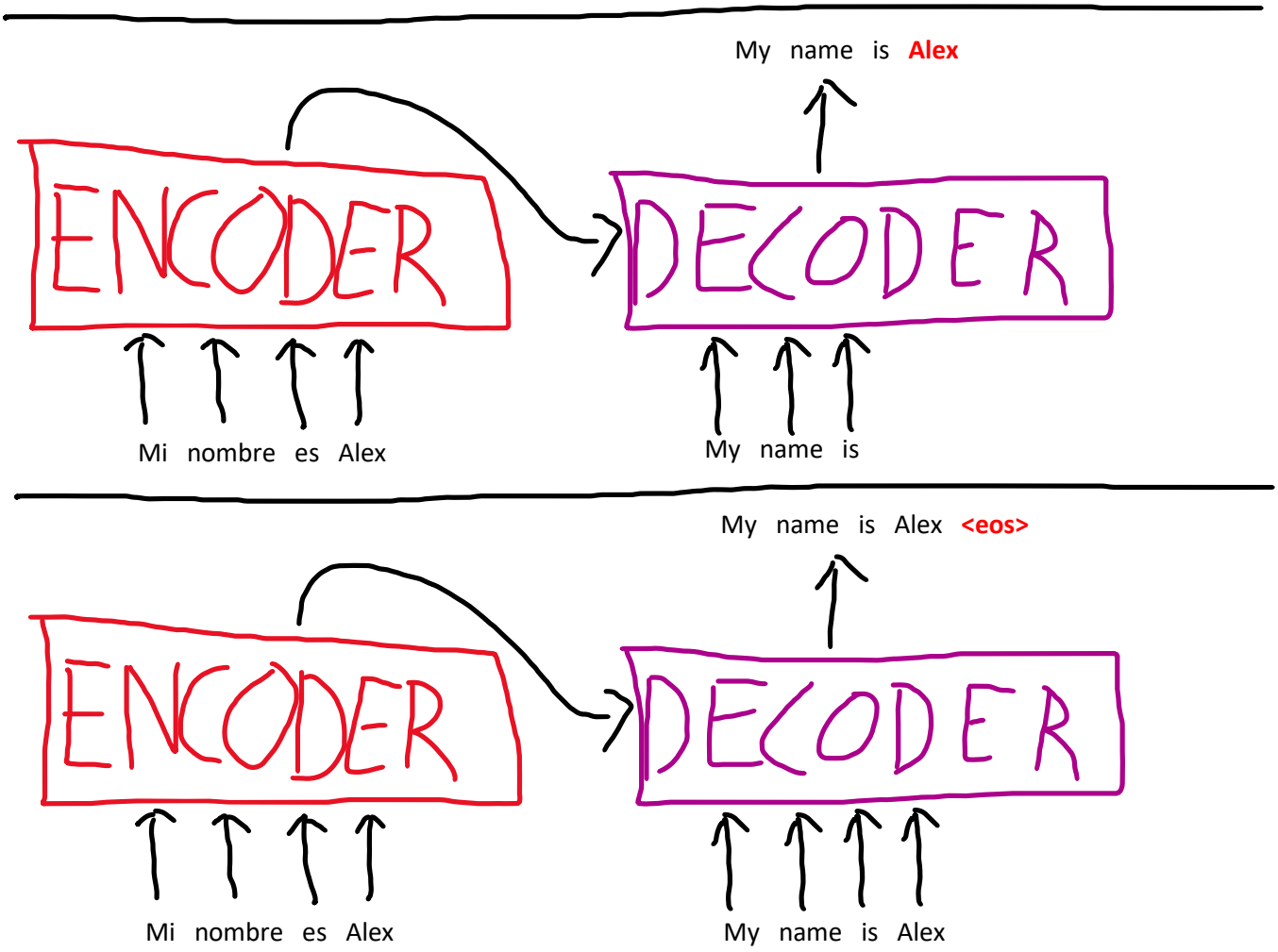
Greedy Decoding: Always choose the highest probability word.

Beam Search: Choose N highest probability words and repeat for each word.

4. Example

The model is run until the end of line character is return:




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