UNIVERSIDAD NACIONAL DE COLOMBIA

Attention Mechanisms and Transformer Architecture

Sebastian López ·, Santiago Pineda, · Rafael Mejia , · Andrés Álvarez

Digital Signal Processing and Control Group - (GCPDS)
Universidad Nacional de Colombia
Manizales, Colombia
June 2023

Contenido

Queries, Keys and Values

consider a database $\mathbf{D}$ of m tuples of keys and values:

$$\mathbf{D} = \{(\mathbf{k_1}, \mathbf{v_1}), ..., (\mathbf{k_m}, \mathbf{v_m})\} \tag{1}$$

and $\mathbf{q}$ is a query that we do on the database, we can define the attention of $\mathbf{q}$ on $\mathbf{D}$ throught the **attention pooling operation**:

$$Attention(\mathbf{q}, \mathbf{D}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k_i})\mathbf{v_i} \in \mathbb{R}^v \tag{2}$$

where:

- $\mathbf{q} \in \mathbb{R}^d$
- $\mathbf{k_i} \in \mathbb{R}^d$
- $\mathbf{v_i} \in \mathbb{R}^v$
- $\alpha(\mathbf{q}, \mathbf{k_i}) \in \mathbb{R}(i = 1, ..., m)$ is the attention weights or the attention function

$$Attention \iff more\,weight$$

Special Cases of the attention weights

UNIVERSIDAD NACIONAL DE COLOMBIA

- $\alpha(\mathbf{q}, \mathbf{k_i})$ are not negative.
- One of the weights $\alpha(\mathbf{q}, \mathbf{k_i})$ is 1, while all others are 0. This is akin to a traditional database query.
- All weights are iqual, $(\mathbf{q}, \mathbf{k_i}) = \frac{1}{m}$.This amounts to averaging across the entire database, also called average pooling in deep learning.
- The weights $\alpha(\mathbf{q}, \mathbf{k_i})$ form a convex combination, that is $\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k_i}) = 1$ and $\alpha(\mathbf{q}, \mathbf{k_i}) \geq 0$ for all i. This is the most common setting in deep learning.

Special Cases of the attention weights

to ensure the most common weight configuration, we need to normalize:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = \frac{\alpha(\mathbf{q}, \mathbf{k_i})}{\sum_j \alpha(\mathbf{q}, \mathbf{k_i})} \tag{3}$$

and to guarantee the non-negativity of the weights we can resort to exponentiation; therefore, to achieve this configuration, we can resort to the SOFTMAX activation function:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = \frac{exp(\alpha(\mathbf{q}, \mathbf{k_i}))}{\sum_j exp(\alpha(\mathbf{q}, \mathbf{k_i}))} \tag{4}$$

¡SOFTMAX has desirable properties for a model: It is differentiable and its gradient never disappears!

Attention Mechanism/Attention Pooling

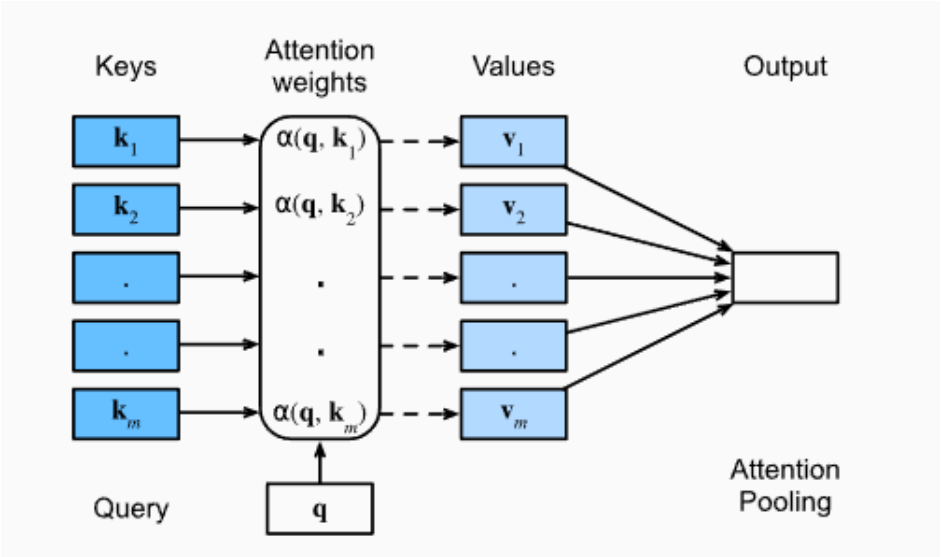

Figure: The attention mechanism computes a linear combination over values $\mathbf{v_i}$ via attention pooling, where weights are derived according to the compatibility between a query $\mathbf{q}$ and keys $\mathbf{k_i}$

¡This is the attention mechanism most used currently in the transformer architectures, but is not the unique!

Attention Scooring Functions

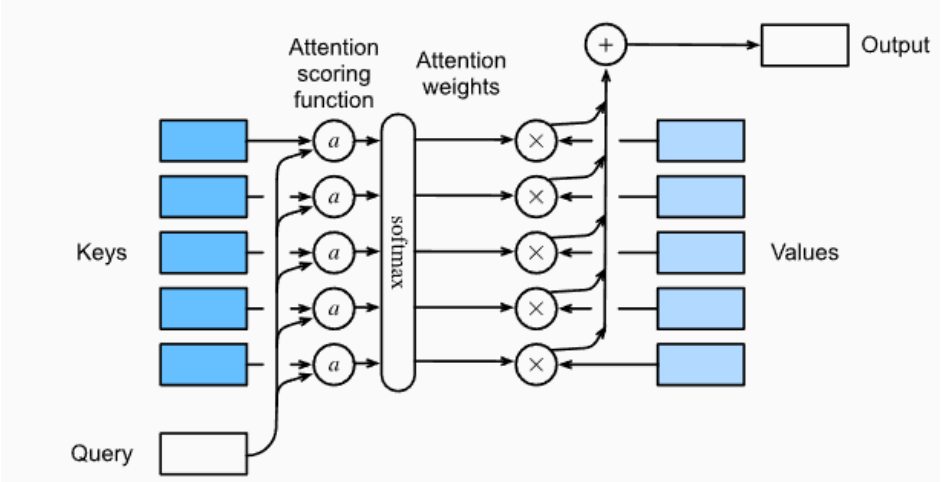

Figure: Computing the output of attention pooling as a weighted average of values, where weights are computed with the attention scoring function and the softmax operation.

Some Attention Functions:

- Gaussian Attention Function: $\alpha(\mathbf{q}, \mathbf{k_i}) = exp(-\frac{1}{2}\|\mathbf{q} - \mathbf{k_i}\|_2^2)$
- Boxcar Attention Function:$\alpha(\mathbf{q}, \mathbf{k_i}) = 1$ if $\|\mathbf{q} - \mathbf{k_i}\|_2 \leq 1$
- Epanechikov Attention Function:$\alpha(\mathbf{q}, \mathbf{k_i}) = max(0, 1 - \|\mathbf{q} - \mathbf{k_i}\|_2)$

Dot Product Attention Function

Previous attention Functions are not used in transformer architectures, but the **Dot Product Attention Function** is a very used attention function used in transformer architectures. Let's start from the gaussian attention function without exponentiation (To reduce computational cost):

$$\alpha(\mathbf{q}, \mathbf{k_i}) = -\frac{1}{2}\|\mathbf{q} - \mathbf{k_i}\|_2^2 = \mathbf{q}^T\mathbf{k_i} - \frac{1}{2}\|\mathbf{k_i}\|_2^2 - \frac{1}{2}\|\mathbf{q}\|_2^2 \qquad (5)$$

Note that both batch and layer normalization lead to activations that have well-bounded, and often constant norms $\|\mathbf{k_i}\|_2 \approx constant$ and note that the last term depends on $\mathbf{q}$ only, as such it is identical for all $(\mathbf{q}, \mathbf{k_i})$ pairs. Then we can drop these terms from the definition of α without any major change in the outcome.

Dot Product Attention Function

Assumming that all elements of the query $\mathbf{q} \in R^d$ and the key $\mathbf{k_i} \in R^d$ are independent and identically distributed random variables with mean zero and variance one, then the dot product between both vectors has mean zero and variance d, so that to ensure that the variance of the dot product is one, we change the scale of the dot product to $\frac{1}{\sqrt{d}}$. See here this proof.

$$\alpha(\mathbf{q}, \mathbf{k_i}) = \frac{\mathbf{q}^T \mathbf{k_i}}{\sqrt{d}} \in \mathbb{R} \tag{6}$$

To ensure the most common setting of weights, we use the SOFTMAX activation function:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = SOFTMAX(\frac{\mathbf{q}^T \mathbf{k_i}}{\sqrt{d}}) = \frac{exp(\frac{\mathbf{q}^T \mathbf{k_i}}{\sqrt{d}})}{\sum_j exp(\frac{\mathbf{q}^T \mathbf{k_i}}{\sqrt{d}})} \tag{7}$$

¡Attention Function very used in transformer architectures!

Masked Softmax Operation

it is important that the attention mechanism knows how to handle sequences with variable lengths (common for NLP), the masked softmax operation is a attetnion mechanism that allows to handle this. Ex:

```
Dive  into  Deep    Learning
Learn to    code    <blank>
Hello world <blank> <blank>
```

Figure: Sequences of Different Lengths

the operation consists of limiting the linear combination of the attention mechanism so that it does not take blank spaces into account:

Masked Softmax Operation

$$\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k_i})\mathbf{v_i} \rightarrow \sum_{i=1}^{l} \alpha(\mathbf{q}, \mathbf{k_i})\mathbf{v_i} \tag{8}$$

where $l \leq m$

Actually, the implementation cheats ever so slightly by setting the values to zero $\mathbf{v_i} = 0$ for $i > l$

Vectors $\mathbf{q}$ and $\mathbf{k_i}$ with different lengths

UNIVERSIDAD NACIONAL DE COLOMBIA

Given a query $\mathbf{q} \in \mathbb{R}^q$ and a key $\mathbf{k_i} \in \mathbb{R}^k$

we can address this by replacing $\mathbf{q}^T\mathbf{k_i}$ with $\mathbf{q}^T\mathbf{M}\mathbf{k_i}$

where $\mathbf{M} \in \mathbb{R}^{q \times k}$ is a suitably chosen matrix to translate between both spaces

we can also address this by making use of the **Additive Attention Function**:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = \mathbf{w_v} tanh(\mathbf{W_q q} + \mathbf{W_k k_i}) \in \mathbb{R} \tag{9}$$

Vectors $\mathbf{q}$ and $\mathbf{k_i}$ with different lengths

Where $\mathbf{W_q} \in \mathbb{R}^{h \times q}$, $\mathbf{W_k} \in \mathbb{R}^{h \times k}$ and $\mathbf{w_v} \in \mathbb{R}^{h}$ are parameters learnable by the model (**Backpropagation**).

To ensure the most common setting of weights, we use the SOFTMAX activation function:

$$\alpha(\mathbf{q}, \mathbf{k_i}) = SOFTMAX(\mathbf{w_v} tanh(\mathbf{W_q q} + \mathbf{W_k k_i})) \in \mathbb{R} \quad (10)$$

Multiples queries over $\mathbf{D}$

So far we have studied attention mechanisms for when we have a single query $\mathbf{q}$ over m key-value tuples: $\mathbf{q} \in \mathbb{R}^d$, $\mathbf{K} \in \mathbb{R}^{m \times d}$ and $\mathbf{V} \in \mathbb{R}^{m \times v}$

if we have n queries over m key-value tuples: $\mathbf{Q} \in \mathbb{R}^{n \times d}$, $\mathbf{K} \in \mathbb{R}^{m \times d}$ and $\mathbf{V} \in \mathbb{R}^{m \times v}$. Then we can define the dot product attention function as follows:

$$SOFTMAX(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v} \tag{11}$$

¡now we have a matrix of weights: For every query we have a attention weight for every element of the value vector!

¿Where do the queries, keys and values come from?

The input $\mathbf{X} \in \mathbb{R}^{n \times d}$ is transformed into the query matrix $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, the key matrix $\mathbf{K} \in \mathbb{R}^{n \times d_k}$, and the value matrix $\mathbf{V} \in \mathbb{R}^{n \times d_v}$ via three linear transformations:

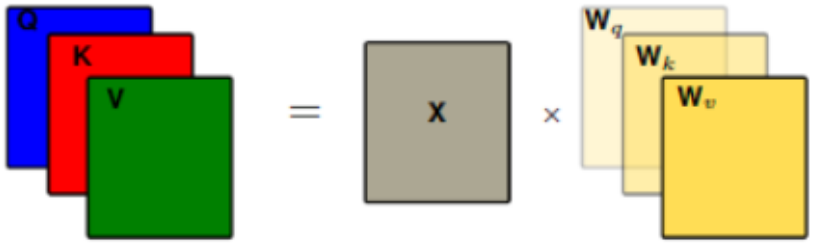

Figure: Linear Transformations

¿Where do the queries, keys and values come from?

UNIVERSIDAD NACIONAL DE COLOMBIA

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_{\mathbf{Q}} \tag{12}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_{\mathbf{K}} \tag{13}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_{\mathbf{V}} \tag{14}$$

where $\mathbf{W}_{\mathbf{Q}} \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}_{\mathbf{K}} \in \mathbb{R}^{d \times d_k}$ and $\mathbf{W}_{\mathbf{V}} \in \mathbb{R}^{d \times d_v}$ are parameters learnable by the model (**Backpropagation**).

Multihead Attention

UNIVERSIDAD NACIONAL DE COLOMBIA

The attention mechanism jointly uses h different representation subspaces of the matrices Q, K, V; since this will allow that in each representation subspace different patterns can be discovered among the data. These h representation subspaces are obtained from transforming with h linear projections **independently** learned by the model (**Backpropagation**). Then, the h linear projections feed the attention mechanism in parallel and in the end the results are concatenated and transformed with another linear projection.

Multihead Attention

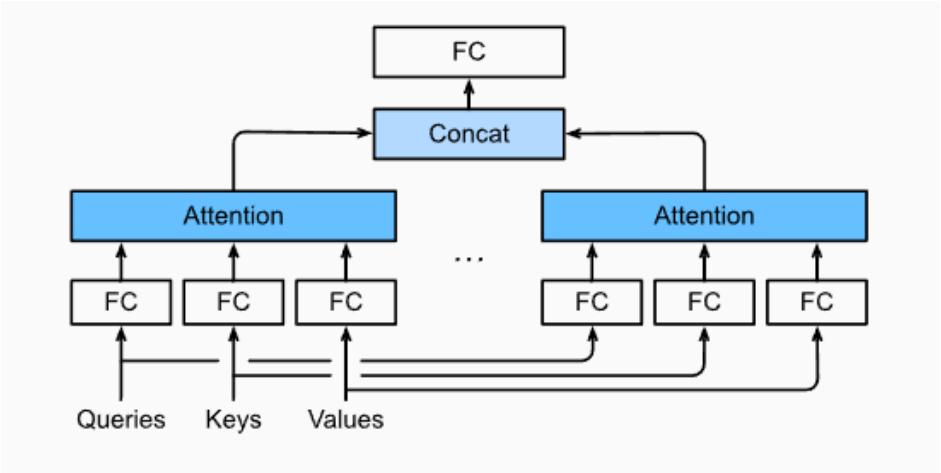

Figure: Multi-head attention, where multiple heads are concatenated then linearly transformed

¡Here the power of transformers:They are highly parallelizable!

Multihead Attention

Given a query $\mathbf{q} \in \mathbb{R}^{d_q}$, a key $\mathbf{k_i} \in \mathbb{R}^{d_k}$ and a value $\mathbf{v_i} \in \mathbb{R}^{d_v}$, each attention head $h_i (i = 1, ..., h)$ is calculated as:

$$h_i = f(\mathbf{W_i}^{(q)}\mathbf{q}, \mathbf{W_i}^{(k)}\mathbf{k_i}, \mathbf{W_i}^{(v)}\mathbf{v_i}) \in \mathbb{R}^{p_v} \tag{15}$$

where $\mathbf{W_i}^{(q)} \in \mathbb{R}^{p_q \times d_q}$, $\mathbf{W_i}^{(k)} \in \mathbb{R}^{p_k \times d_k}$ and $\mathbf{W_i}^{(v)} \in \mathbb{R}^{p_v \times d_v}$ are parameters learnable by the model (**Backpropagation**).

$f(.)$ is the attention pooling function, and the attention function can be for example the dot product attention function.

Multihead Attention

The output of the multihead attention mechanism is calculated as:

$$\mathbf{W_o} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_h \end{bmatrix} \in \mathbb{R}^{p_o} \tag{16}$$

where $\mathbf{W_o} \in \mathbb{R}^{p_o \times h p_v}$ is the parameter learnable by the model (**Backpropagation**).

¡Conclusion: This design allows each head to serve different parts of the input!

Self Attention

UNIVERSIDAD NACIONAL DE COLOMBIA

Self attention is a mechanism that allows capturing relationships between elements of a sequence (for example, between words in a text) but also between elements of non-sequential data (for example, between pixels in an image). Well, self attention allows finding the attention of each element over all the others.

Given an input sequence of n elements $\mathbf{x_1}, ..., \mathbf{x_n}$ where $\mathbf{x_i} \in \mathbb{R}^{\mathbf{d}} (1 \leq i \leq n)$. Then, for every element is calculated $\mathbf{q_i} \in \mathbb{R}^{d_k}, \mathbf{ki} \in \mathbb{R}^{d_k}, \mathbf{v_i} \in \mathbb{R}^{d_v}$:

Self Attention

UNIVERSIDAD NACIONAL DE COLOMBIA

$$\mathbf{q_i} = \mathbf{x_i}\mathbf{W_{Q_i}} \tag{17}$$

$$\mathbf{k_i} = \mathbf{x_i}\mathbf{W_{K_i}} \tag{18}$$

$$\mathbf{v_i} = \mathbf{x_i}\mathbf{W_{V_i}} \tag{19}$$

where $\mathbf{W_Q} \in \mathbb{R}^{d \times d_k}$, $\mathbf{W_K} \in \mathbb{R}^{d \times d_k}$ and $\mathbf{W_V} \in \mathbb{R}^{d \times d_v}$ are parameters learnable by the model (**Backpropagation**).

After is calculated the attention weights between the element at position i and all other elements:

$$\alpha(\mathbf{q_i}, \mathbf{k_j}) = SOFTMAX(\frac{\mathbf{q_i}^T \mathbf{k_j}}{\sqrt{d}}) \in \mathbb{R} \tag{20}$$

Self Attention

UNIVERSIDAD NACIONAL DE COLOMBIA

as we can see this is calculated using the query of the element at position i and the keys of the other elements.
After is calculated the pooling attention of the element at position i over all other elements:

$$Attention(\mathbf{q_i}, \mathbf{D_j}) = \sum_m \alpha(\mathbf{q_i}, \mathbf{k_j})\mathbf{v_j} \in \mathbb{R}^v \tag{21}$$

The same procedure is repeated over all the other elements and with all the other elements.

Comparison between CNN, RNN and Self Attention

UNIVERSIDAD NACIONAL DE COLOMBIA

	Computational Cost	Sequential Operations	Maximum Route Length
CNN	Depends on the size of the input data, the number of convolutional layers, and the number of filters.	Multiple filters on different parts of the image at the same time **(parallelizable).**	There is no notion of maximum paths.
RNN	Depends on the length of the input sequence and the number of time steps.	All calculations are sequential **(not parallelizable).**	Long-term memory loss: Limitation to capture long-term dependencies.
Self Attention	Depends on the size of the input.	The attention of each element on the others can be calculated simultaneously **(Parallelizable).**	Capture long-term dependencies; since each element attends to all the others at the same time.

Tabla 1: Comparación entre CNN, RNN y Self Attention.

¡Self-attention mechanisms meet two desirable requirements: They encode long-term sequences and are fast (because they are parallelizable)!

Comparison between CNN, RNN and Self Attention

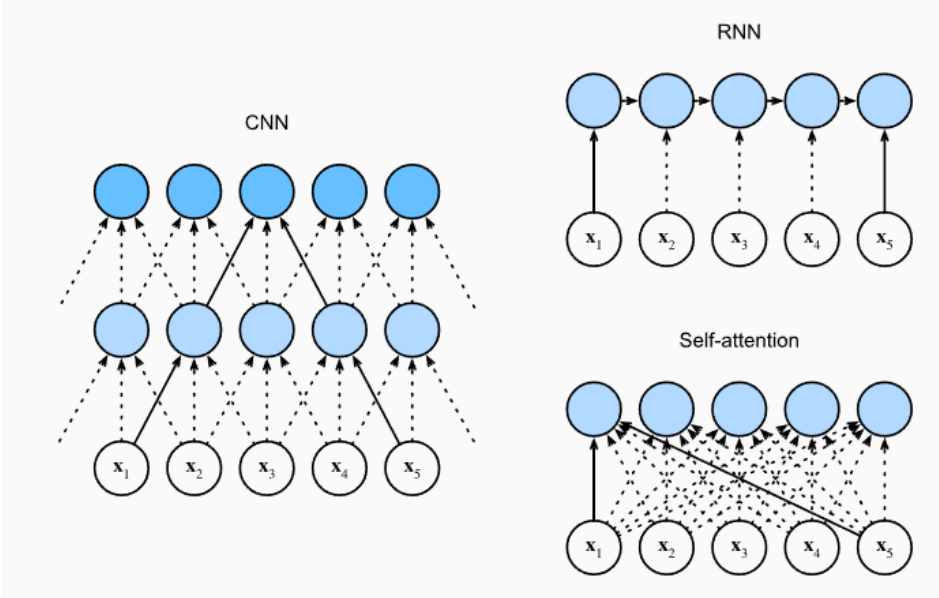

Figure: Comparing CNN (padding tokens are omitted), RNN, and self-attention architectures.

$$RNN \iff Transformer\ Architecture$$

Positional Encoding

UNIVERSIDAD NACIONAL DE COLOMBIA

Now, before passing the input $\mathbf{X} \in \mathbb{R}^{n \times d}$ through a self-attention mechanism, its respective positional information must be added to each element $\mathbf{x_i} \in \mathbb{R}^d$; since self-attention is a parallel mechanism, the positional information of the elements is lost.

Positional encoding consists of adding to each element a vector that encodes both positional information and frequency information. This positional coding can be learned by the model (backpropagation) or it can be fixed a priori using sine and cosine functions.

Positional Encoding

We have some input data $\mathbf{X} \in \mathbb{R}^{n \times d}$, then the output of the positional encoding is $\mathbf{X} + \mathbf{P}$ where $\mathbf{P} \in \mathbb{R}^{n \times d}$ is a positional embedding matrix.
Then, the positional embedding for the i-th element in every column(j) of the element is:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \in [-1, 1] \tag{22}$$

,

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \in [-1, 1] \tag{23}$$

where $p_{i,2j}$ is for pairs columns and $p_{i,2j+1}$ is for uninpairs columns.
$NLP \rightarrow Words\ Position\ (Temporal\ Information)$
$CV \rightarrow Pixels\ Position\ (Spatial\ Information)$

Transformer Architecture

UNIVERSIDAD NACIONAL DE COLOMBIA

As we have seen, self attention allows both parallel computation (fast) and long-term relationship coding. So it is attractive to design deep architectures using self-attention (This is a transformer).

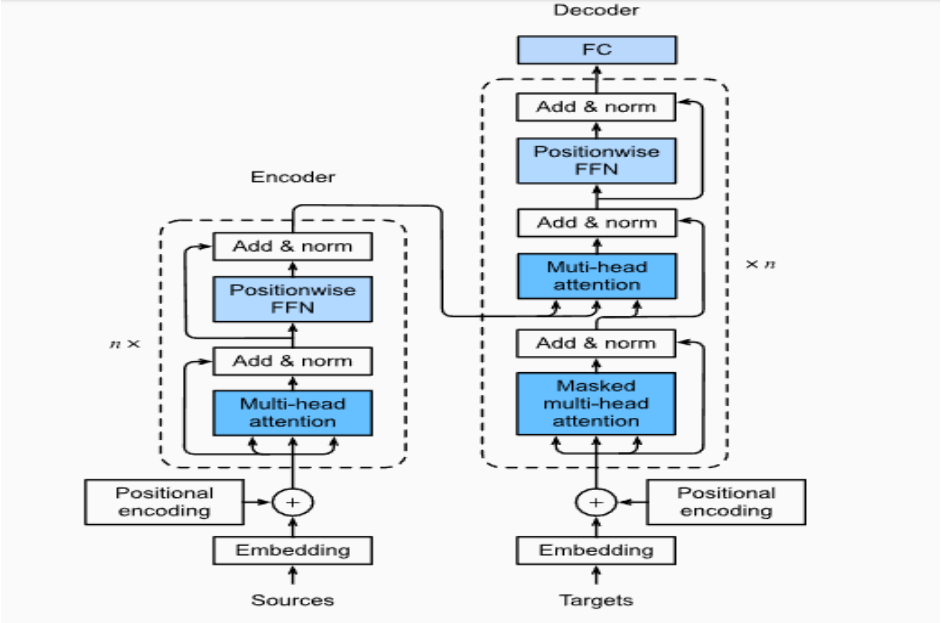

Figure: The Transformer Architecture.

Positional Encoding

The input (source) and output (target) data are passed through a positional encoding before entering the encoder and decoder respectively; since they will enter self attention mechanisms.

$$\mathbf{X}' = \mathbf{X} + \mathbf{P_x} \in \mathbb{R}^{\mathbf{n}\times\mathbf{d}} \tag{24}$$

$$\mathbf{Y}' = \mathbf{Y} + \mathbf{P_y} \in \mathbb{R}^{\mathbf{n}\times\mathbf{d}} \tag{25}$$

where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the input data, $\mathbf{Y} \in \mathbb{R}^{n\times d}$ is the output data and $\mathbf{P_x} \in \mathbb{R}^{n\times d}$, $\mathbf{P_y} \in \mathbb{R}^{n\times d}$ are the respective positional embedding matrices.

Encoder

The encoder is a stack of n identical layers, which have two sub layers: The first is a multi-headed self attention pooling and the second is a positionwise feed-forward network (FFN).

The FFN is in charge of introducing non-linearity to the model (using a Relu activation function); as this allows the model to capture characteristics and patterns of each position in the sequence.

$$FFN(x) = max(0, \mathbf{x}\mathbf{W_1} + b_1)\mathbf{W_2} + b_2 \tag{26}$$

Where x is the input to FFN and $\mathbf{W_1}, \mathbf{W_2}, b_1, b_2$ are the parameters learned by the model (Backpropagation).

Encoder

A residual connection and a layer normalization are applied to both sublayers.

The residual connection consists of adding a direct connection (sum) between the input and output of the sublayer ; as this will allow the original input to be preserved throughout the model and flow unhindered through the model.

$$\mathbf{Y} = \mathbf{X} + sublayer(\mathbf{X}) \in \mathbb{R}^{n \times d} \tag{27}$$

Then the layered normalization is applied after each residual connection, this in order to reduce the variance of the activations and help the gradient to converge faster.

Decoder

The decoder is also a stack of n identical layers with residual connections and layer normalizations. But here, each layer has a third sublayer called encoder-decoder attention, which allows communication between encoder and decoder; as it allows the decoder to service the encoder outputs, allowing the relevant information to flow from the encoder to the decoder. Here the queries are from the output of the previous layer of the decoder and the keys and values are from the output of the encoder.

UNIVERSIDAD NACIONAL DE COLOMBIA

Thanks!