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

UNIVERSIDAD NACIONAL DE COLOMBIA

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

- $\alpha(\mathbf{q}, \mathbf{k_i})$ 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

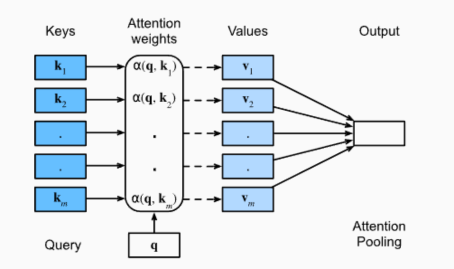

Figure: The attention mechanism computes a linear combination over values $\mathbf{v_i}$ via attention pooling, where weights are derived according to the compatibility between a query $\mathbf{q}$ and keys $\mathbf{k_i}$

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

Attention Scoring Functions

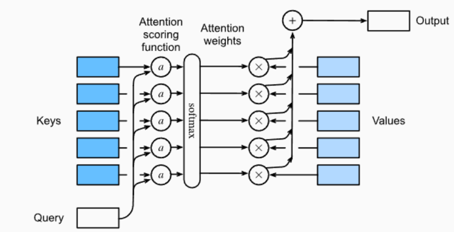

Figure: Computing the output of attention pooling as a weighted average of values, where weights are computed with the attention scoring function and the softmax operation.

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

UNIVERSIDAD NACIONAL DE COLOMBIA

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:

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

¡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

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 **Adictive Attention Function**:

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

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:

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

¡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:

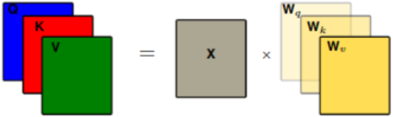

Figure: Linear Transformations

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

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

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

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

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

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

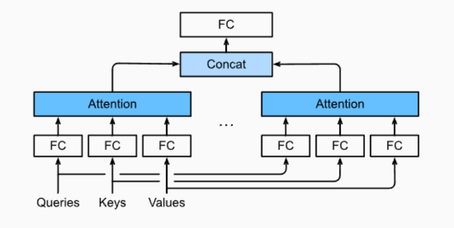

Figure: Multi-head attention, where multiple heads are concatenated then linearly transformed

¡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!

Thanks!