Attention is about 1/3 of what you need

ElMan23 following 3Blue1Brown

July 27, 2024

Attention

Imagine that we want to create a model that is able to produce a sentence by predicting for each work the most probable follower. This model should clearly pay attention to the context.

For the sake of concreteness, let's analyse a sentence like:

> A fluffy blue creature roamed the verdant forest.

The first step is to represent the sentence in some mathematical form. For this aim, we *tokenise* the sentence.

Loosely speaking, we can imagine a token as a word in the sentence. This is rather imprecise, for example the word "tokenisation" is composed by two tokens ("token" and "isation"), but for our little example there is no bad in this incorrect simplifying assumption.

We start therefore by dividing the sequence in tokens. In our case, we would get:

A	fluffy	blue	creature	roamed	the	verdant	forest

Each square represents a token.

Thus we have a set of tokens $\{t_i \mid i = 1, ..., 8\}$.

Now we *embed* the tokens in a vector space. Specifically, in our case, the *dimension* of the vector space may be quite big, for example 12288: this is defined by the number of tokens. Nevertheless, assume that our embedding vector space is E. Then we can associate to each token t_i, for $i = 1, ..., 8$, a certain vector $\vec{e}_i \in E$:

$$t_i \mapsto \vec{e}_i \in E.$$

Queries

A query is a map $W_Q : \vec{e}_i \mapsto W_Q\vec{e}_i$. We write:

$$\vec{q}_i = W_Q\vec{e}_i.$$

The map sends the vectors in E to vectors in another space, F, which has lower dimension (in the case where $dim(E) = 12288$, we can think of $dim(F) = 128$).

Keys

A key is a map $W_k : \vec{e}_i \mapsto W_k\vec{e}_i$. We write:

$$\vec{k}_i = W_k\vec{e}_i.$$

The map sends the vectors in E to vectors into F, which has lower dimension (in the case where $dim(E) = 12288$, we can think of $dim(F) = 128$).

We can now take the dot products:

$$\vec{k}_i \cdot \vec{q}_j$$

which define a matrix as i and j vary.

When $\vec{k}_i$ and $\vec{q}_j$ are close, then the dot product $\vec{k}_i \cdot \vec{q}_j$ is big.

In a sense, the query $\vec{q}_j$can be thought as an operator looking for something relevant in the context for the embedded token $\vec{e}_j$, while the key $\vec{k}_i$ is the vector to which we apply the query. Thus $\vec{k}_i \cdot \vec{q}_j$ looks for some relevant feature for $\vec{e}_j$ contextualised in $\vec{e}_i$.

The matrices W_Q and W_k representing the application are **learned from the data**.

Recap

Queries:

$$W_Q : \vec{e}_i \mapsto \vec{q}_i = W_Q\vec{e}_i$$
$$W_Q : 12288D \to 128D$$

Keys:

$$W_k : \vec{e}_i \mapsto \vec{k}_i = W_k\vec{e}_i$$
$$W_k : 12288D \to 128D$$

Matrix:

$$\left[\vec{k}_i \cdot \vec{q}_j\right]$$

Example

Let's consider the tokens: fluffy | blue | creature

These are embedded to $\vec{e}_2, \vec{e}_3, \vec{e}_4$.

We compute:

$$\vec{q}_2 = W_Q\vec{e}_2,\ \vec{q}_3 = W_Q\vec{e}_3,\ \vec{q}_4 = W_Q\vec{e}_4,$$

$$\vec{k}_2 = W_k\vec{e}_2,\ \vec{k}_3 = W_k\vec{e}_3,\ \vec{k}_4 = W_k\vec{e}_4.$$

Recall that W_Q and W_k will be learned from data; we assume that we have already learned the matrices.

Since "fluffy" and "blue" are relevant in the context for "creature", then we expect that $\vec{k}_2 \cdot \vec{q}_4$ and $\vec{k}_3 \cdot \vec{q}_4$ will be large and positive.

We say that the embeddings o "fluffy" and "blue" attend to the embedding of "creature".

Normalization

Consider the matrix

$$\left[\vec{k}_i \cdot \vec{q}_j\right]$$

In general $\vec{k}_i \cdot \vec{q}_j \in (-\infty, \infty)$.

We would like $\vec{k}_i \cdot \vec{q}_j \in [0, 1]$ and

$$\sum_i \vec{k}_i \cdot \vec{q}_j = 1$$

i.e., each column adds up to 1.

We apply softmax.

In general, for $\vec{x} = (x_1, ..., x_n)$, we have:

$$\left(\text{softmax}\left(\vec{x}\right)\right)_i = \frac{\exp\left(x_i\right)}{\sum_j \exp\left(x_j\right)}.$$

Thus, we apply softmax to each column, normalizing it.

Since we want to predict the next word based on the previous ones, we don't want the later symbols to influence the previous ones.

Thus, the matrix

$$\vec{k}_i \cdot \vec{q}_j$$

should be upper triangular, i.e. each entry below the main diagonal should be null.

Since we are applying softmax function to each column, we don't actually set the values of $\vec{k}_i \cdot \vec{q}_j$ below the diagonal to 0, but we set them to $-\infty$, so that the softmax will convert them to 0 keeping at the same time the normalisation of the column and the sum of the entries equal to 1.

This is called **masking**.

Compact form

We can represent attention in a compact form.

Represent the $\vec{q}_i$ in a matrix Q and the $\vec{k}_i$ in a matrix K. The symbol QK^T is a compact way to represent the matrix

$$\left[\vec{k}_i \cdot \vec{q}_j\right]$$

Thus:

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

The division by $\sqrt{d_k}$ is applied for numerical stability. The softmax is applied column by column.

The matrix V is the value matrix.

Values

We compute the values
$vecv_i$ as follows.

We use a new matrix W_V, again learned from the data. Thus:

$$W_V : \vec{e}_i \mapsto \vec{v}_i = W_V \vec{e}_i.$$

We represent by $\sigma\left(\vec{k}_i \cdot \vec{q}_j\right)$ the i, j entry result of applying softmax to the column.

For each value $\vec{v}_i$, we compute moreover:

$$\sigma\left(\vec{k}_i \cdot \vec{q}_j\right)\vec{v}_i.$$

This is a vector quantity $\vec{v}_i$ multiplied by the scalar $\sigma\left(\vec{k}_i \cdot \vec{q}_j\right)$, thus a vector. This is the generic entry in

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Finally, we sum the column:

$$\Delta\vec{e}_j = \sum_i \sigma\left(\vec{k}_i \cdot \vec{q}_j\right)\vec{v}_i.$$

This is the displacement we will apply to $\vec{e}_j$ to update it using the context:

$$\vec{e}_j \mapsto \vec{e}_j\,' = \vec{e}_j + \Delta\vec{e}_j$$

This is a single **head of attention**.

The parameters: W_Q, W_k, W_V, queries, keys and values.

In general

$$W_Q, W_k \in M(128, 12288)$$

and

$$W_V \in M(12288, 12288),$$

but we can constrain W_V to be a low rank transformation, i.e. there exist

$$V_d \in M(128, 12288),\ V_u \in M(12288, 128)$$

such that

$$W_V = V_u V_d.$$

A single head of attention focuses on a single trait of the context. Still, the context can influence the meaning in many ways. Thus, we need **multi-headed attention**.

For example, GPT3 uses 96 attention heads inside each block.

Each head of attention produces a proposed change $\Delta\vec{e}_j j^{(j)}$. The updated vector will be computed using the sum:

$$\vec{e_i} \mapsto \vec{e}_i + \sum_j \Delta\vec{e}_i^{(j)}.$$

Many heads of attention means many ways in which the context can influence the meaning (i.e. the *direction* of the final vector in the space).

Output

In general V_u, V_d are represented by a single **output matrix**

$$\left[V_u^{(1)} \cdots V_u^{(n)}\right]$$

and the $V_d^{(j)}$ are called the **values**.

Architecture

We stack attention blocks followed by multilayer perceptrons.

The majority of the parameters come from multilayer perceptrons; hence the title: just about 1/3 of the total parameters come from attention.

References

[1] 3Blue1Brown, *YouTube*, "Attention in transformers, visually explained - Chapter 6, Deep Learning". https://www.youtube.com/watch?v=eMlx5fFNoYc&t=269s. Online; accessed June 27, 2024.