Primitives of transformers

GLyC

October 14, 2025

Embedding with flags

Assume there is a transformer T with embedding size d. It's possible to add flags if we expand the embedding size. Since the dimensions of the embedding will be modified, it is needed to also change the dimensions of all the matrices and the token embedding. That can be done with zeros so it doesn't affect the behavior.

Since the positional embedding from T is just a mapping from $[n_{max}]$ to $\mathbb{R}^d$ it is the perfect place for marking the ith vector. It can be done defining the positional embedding of the transformer with flags as:

$$(\theta'_{TE}(n))_j = \begin{cases} (\theta_{TE}(n))_j & \text{if } j \neq d+1 \\ 1 & \text{if } j = d+1 \text{ and } n = i \\ 0 & \text{if } j = d+1 \text{ and } n \neq i \end{cases}$$

Note that is possible to flag multiple vectors in the same coordinate or to flag with a different value than 1. Also the flagging can be done in any coordinate, not only in the last one.

Add one specific vector to the ith

The i is fixed from the construction of the transformer, it's assumed that is flagged with a 1 in the last coordinate. Therefore all vectors with have 0 in their last coordinate with the exception of the ith which will have a 1.

Lets call v the vector that we want to add. We will do it with one layer of FF. We will construct a layer such that $FF(h_j) = \mathbb{I}(j = i) * v$.

$$W_1 = id \qquad W_2 = \begin{pmatrix} 0 & \dots & 0 & v_1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & v_d \end{pmatrix} \qquad b_1 = b_2 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

If we were to choose the kth coordinate for the flag instead of the last one, then we would have to put v in the kth column.

Make the ith vector zero

With just adding a fixed vector we can transform any vector to a constant one. For that we will first make it zero and then add the one we want. Lets see the first part.

For all x it happens that $(x+\infty)+\infty=\infty$. Since $x \geq -\infty$, it happens that $x+\infty \geq 0$, then $(x+\infty)+\infty \geq \infty$, so $(x+\infty)+\infty=\infty$. Therefore if we want to get zero, it's enough to subtract ∞.

$$h_i \leftarrow \left(\left(h_i + \begin{pmatrix} \infty \\ \vdots \\ \infty \end{pmatrix}\right) + \begin{pmatrix} \infty \\ \vdots \\ \infty \end{pmatrix}\right) + \begin{pmatrix} -\infty \\ \vdots \\ -\infty \end{pmatrix}$$

ATTN y FF controlling which coordinates are affected

Later we will see that we want to go trough layers of attention and feed forward ignoring some coordinates. That means without taking them into consideration and without affecting them in the output.

For example, if we want to ignore the ith coordinate trough a layer of attention, it's enough to put zeros in the ith column and row of W_Q, W_K, W_V, W_O. This can be done with as many coordinates as desired. Note that because the ith row of W_O is zero, the output vector of the ATTN layer will have a zero in the ith coordinate. This is the point that makes it possible to not just ignore the value of the ith coordinate but to not affect it. This happens because the result of the attention is added to the original vector.

The same applies in the feed forward layer, putting zeros in the ith column and row of the matrix and vectors of the FF makes it ignore the value of the ith coordinate. Also in the result of the FF will appear a zero in the ith coordinate.

Linear transformations

We define two ways of applying linear transformations to an specific vector. Both have the effect of corrupting every vector except the one with the result. The first one saves the answer in a different vector to the right of the one receiving the transformation. Note that this requires adding a extra character at the end (so we get and extra vector) if we want to apply a transformation to the last vector, which we will need for the normalizations. This is uncomfortable for doing CoT. Thats why we also introduce the second way of applying linear transformation which saves the answer in the same vector, but requires to double the embedding size.

Linear transformation storing to the right

In this section we will see how to apply a linear transformation to one of the vectors and save the output in another more to the right, in other words: $h_j^{l+c} = Mh_i^l$ for some $i < j$ previously chosen.

Note that after this procedure all the vector will have their values corrupted with the exception of h_j.

First we will make $h_j = 0$ at the coordinates we are interested (this can be achieved with the previous sections). Then with an ATTN layer we will add Mh_i.

Using the previous sections we can assume that every vector's last two coordinates are $(0, 1)$, except h_i's which are $(1, 0)$. These numbers would have been set by the encoding and preserved trough the previous layers.

Assuming this we will construct W_K and W_Q such that:

$$\langle q_j, k_{i'} \rangle = \begin{cases} -\infty & \text{if } i' \neq i \\ 0 & \text{if } i' = i \end{cases}$$

Note that the W_Q and W_K that we are looking for are the following:

$$W_Q = \begin{pmatrix} 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & -\infty \end{pmatrix} \qquad W_K = \begin{pmatrix} 0 & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix}$$

since

$$q_j = W_Q\ h_j = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -\infty \end{pmatrix} \qquad k_{i'} = W_K\ h_{i'} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (h_{i'})_d \end{pmatrix}$$

so $\langle q_j, k_{i'} \rangle = -\infty * (h_{i'})_d$

Taking $W_O = M$ y $W_V = id$ we get what we were looking for.

This works because $e^{-\infty} = 0$ and $softmax(-\infty, \dots, -\infty, 0) = (0, \dots, 0, 1)$, which makes $(s_j)_{i'} = \mathbb{I}(i' = i)$. In summary:

$$W_O \sum_{i'=0}^{n} (s_j)_{i'} v_{i'} = W_O \sum_{i'=0}^{j} (s_j)_{i'} v_{i'} = W_O \sum_{i'=0}^{j} (s_j)_{i'} h_{i'} =$$

$$= W_O \sum_{i'=0}^{j} \mathbb{I}(i' = i) h_{i'} = W_O\ h_i$$

Linear transformation doubling the embedding size

As it was said before, if we want to store the result of the linear transformation in the same vector (because of the error given by the finite representation)[1] we will have to double the embedding size.

The linear transformation will be realized trough two layers of ATTN. The first layer will store the answer of the linear transformation in the second half of the embedding and overwrite the first one with zeros. Then, the second layer will swap the two parts so the answer its finally stored in the first part of the embedding. It will also overwrite the second half with zeros so it can be reused.

First, its needed to extend all previous matrices and embeddings from dimension d to dimension $2 * d + 1$. This has to be done with zeros in everyplace with the exception of the positional embedding and the last coordinate of the vector we want to multiply. The plus 1 came from the flag and the times 2 for having space to save the result.

As we assumed before, the vector we will want to multiply by a matrix will be flagged with a 1 in the last coordinate and all the other will have a 0. As seen before the matrices for W_Q and W_K will be such that h_i only attends to its self:

$$W_Q = W_K = \begin{pmatrix} 0 & \dots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & \infty \end{pmatrix}$$

Then,

$$(W_Q h_{i'})_j = (W_K h_{i'})_j = \begin{cases} 0 & \text{if } j \neq 2 * d + 1 \\ 0 & \text{if } j = 2 * d + 1 \wedge i' \neq i \\ \infty & \text{if } j = 2 * d + 1 \wedge i' = i \end{cases}$$

W_V will be the identity and W_O will be:

$$W_O = \begin{pmatrix} -id^{\mathbb{R}^{d \times d}} & 0^{\mathbb{R}^{d \times d}} & 0^{\mathbb{R}^d} \\ M & 0^{\mathbb{R}^{d \times d}} & 0^{\mathbb{R}^d} \\ 0^{\mathbb{R}^{1 \times d}} & 0^{\mathbb{R}^{1 \times d}} & 0^{\mathbb{R}} \end{pmatrix}$$

Where M is the matrix of the linear transformation wanted to apply.

Note than in the ATTN vector, all the values will be 0 except for $\langle q_i, k_i \rangle$ which will be 1. Lets call h_i^1 to the first half of h_i. Then, the output of the ATTN layer of the ith vector will be: $(-h_i^1, M h_i^1, 0)$ which added to h_i gives us $(0^d, M h_i^1, 1)$.

[1] maybe expand this?

In the second layer we only need to swap the halves of h_i. For that we will use the same query key and value matrices but the W_O will be:

$$W_O = \begin{pmatrix} 0 & id^{\mathbb{R}^{d\times d}} & 0^{\mathbb{R}^d} \\ -id^{\mathbb{R}^{d\times d}} & 0^{\mathbb{R}^{d\times d}} & 0^{\mathbb{R}^d} \\ 0^{\mathbb{R}^{1\times d}} & 0^{\mathbb{R}^{1\times d}} & 0^{\mathbb{R}} \end{pmatrix}$$

Then the output in the ith vector of this layer of ATTN will be $(Mh_i, -Mh_i, 0)$ so after adding it with the previous one we get $(Mh_i, 0^{d+1})$. Exactly what we were looking for.