6- transformers

transformers

- neural network architecture that has been replacing LSTMs (RNNs) and CNNs

attention mechanism = layer that allows the model to focus specific parts of input by assigning different weights to different parts of the input.

1- passes more data = instead of summary, all the hidden states information is passed

2- extra step before producing its output

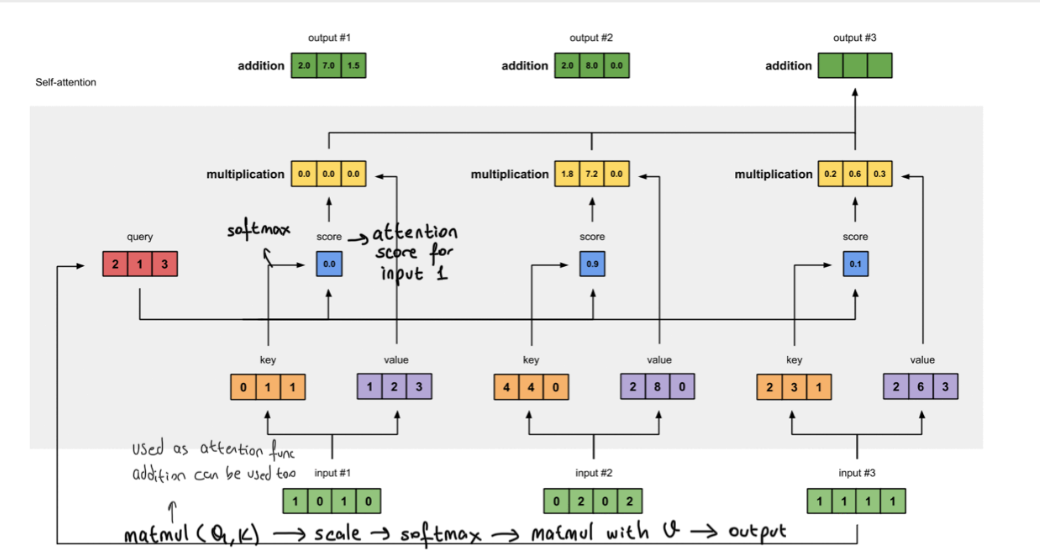
↳ 1. looks at the set of encoder hidden states that it received

↳ 2. gives each hidden state a score

↳ 3. multiply each hidden state by its soft-maxed score

} to focus on the most relevant parts of the input

self-attention = to remove the dependency on sequential processing



the attention mechanism steps

① prepare inputs = 3 inputs (ex: embeddings of 3 words in a sentence)

[1, 0, 1, 0]   [0, 2, 0, 2]   [1, 1, 1, 1]

② initialize weights = to make representations with 2 (bias can be added too)

$$w_{key} = \begin{bmatrix} [0,0], \\ [1,0], \\ [0,1], \\ [1,1] \end{bmatrix} \quad \underbrace{4}_{\text{input size}} \times \underbrace{2}_{\text{output size}} \qquad w_{query} = \begin{bmatrix} [\cdot\ \cdot] \\ \vdots \end{bmatrix} 4\times 2 \qquad w_{value} = 4\times 2$$

learnable parameters

③ derive key, query and value for every input =

$$[1,0,1,0] \begin{bmatrix} [0,0], \\ [1,0], \\ [0,1], \\ [1,1] \end{bmatrix} = [0,2] \rightarrow \text{key for input 1}$$

faster with matmul =

$$\underset{(3\times 4)}{\begin{bmatrix} inp1 \\ inp2 \\ inp3 \end{bmatrix}} \cdot \underset{(4\times 2)}{W_{key}} = \begin{bmatrix} 0 & 2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{matrix} \rightarrow \text{key 1} \\ \rightarrow \text{key 2} \\ \rightarrow \text{key 3} \end{matrix}$$

④ calculate attention score for input 1 = dot product between input 1 query and all keys

$$\underset{\text{query 1}}{[1, 0]} \underset{W_{key}^T}{\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}} = [0\ 1\ 0] \rightarrow \text{3 key, 3 attention scores}$$

dot product attention

$s_t = \text{attention\_function}(y_{i-1}, x_t)$

* there is additive attention too

⑤ calculate softmax = $\frac{e^{s_t}}{\sum_K e^{s_t}}$ softmax([2,4,4]) = [0.0, 0.5, 0.5]

⑥ multiply scores with values = the softmaxed attention scores for each input is multiplied by its corresponding value.

$$\left.\begin{matrix} (0.0) * [1, 2, 3] = [\cdot\ \ \cdot\ \cdot] \\ (0.5) * [\cdot\ \ \cdot\ \ \cdot] = [\cdot\ \cdot\ \cdot] \\ (0.5) * [\cdot\ \ \cdot\ \ \cdot] = [\cdot\ \cdot\ \cdot] \end{matrix}\right\} \text{weight values}$$

⑦ sum weight values to get output 1 = [. . .] + [. . .] + [. . .] = [. . .] → output for input 1

⑧ repeat for input 2 & input 3

- ☆ query and key dimensions must be same (dot product for the score func)
- ☆ value matrix dimension can be different (it will be the output dimensio)
- the order of the words have no influence on each other
- no dependency on the length to compute the similarity between two words

$$f(x_i, x_j) = (W_q \cdot x_i)^T (W_k \cdot x_j) \rightarrow \text{attention score}$$

$$w_{ij} = \frac{\exp(f(x_i, x_j))}{\sum_{k=1}^{n} \exp(f(x_i, x_j))} \rightarrow \text{normalization} \qquad y_i = \sum_{j=1}^{n} w_{ij} (W_v x_j)$$

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

multi-head self attention

↳ each has its own learnable parameters (Q, K, V matrices)

↳ concat the output of N modules

positional encoding = describe the location of an entity in a sequence, so that model can understand the order of the words and their relative positions

- added to input embeddings (feature vectors)
- can be fixed or learnable
- in "attention is all you need", it is based on sine and cos func of different frequencies

transformers

- a nn architecture that contains many self-attention modules
- applies positional encodings to its input and output
- improved accuracy compare to LSTM
- reduced training cost (1/10) compare to LSTM, due to suitibility for parallelism

GPT, BERT, ViT (vision transformer)