Simple variant of self-attention, free from any trainable weights

Sentence = "Your journey starts with one step"

How much attention do we need to pay to each of these words when we look at journey

Embedding → context

"Your" "journey" "starts" ... "step"

$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(T)}$

a_{21} a_{22} a_{23} a_{24}

attention weights: importance of input x_1

$z^{(2)}$ = [4.6, 4.5]

context vector $z^{(2)}$ is computed as a combo of all input vectors weighted w.r.t. input element $x^{(2)}$

input sequence: x consists of T elements

$x^{(1)}, x^{(2)}, \ldots x^{(T)}$

Token embeddings, Token embeddings

3D embedding in example.

Q "calculate a context vector $z^{(2)}$ for each element $x^{(2)}$"

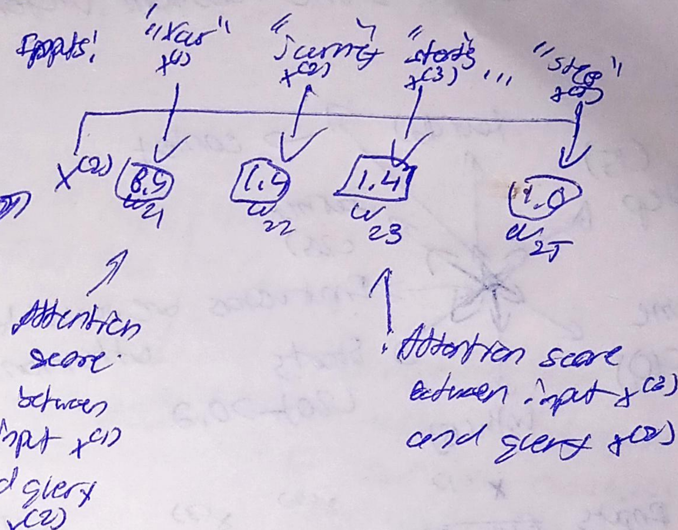
context vector: enriched embedding vector.

③ Focus on 2nd element $x^{(2)}$ (Journey) ← query

↳ corresponding context vector is $z^{(2)}$ which is an embedding which contains information about $x^{(2)}$ and all other ~~Input~~ input elements $x^{(1)}$ to $x^{(T)}$

④ Later, add trainable weights to help LLMs learn to construct these context vectors, so that they are relevant for LLM to generate next token

⑤ Compute intermediate values ω, also referred to as attention scores



* Dot product will be used to calculate similarity between vectors.
  High dot prod → more similar vectors, and higher attention scores
  Lower dot prod → less similar values, and lower attention scores
  ↳ use to find attention score between query vector and input vector

<u>Softmax, why it's better</u>

~~Of Wat~~:

(1, 2, 3, 400) → 0 / → 1

$\frac{1}{406}$ $\frac{2}{406}$ $\frac{3}{406}$ $\frac{400}{406}$

$\frac{1}{406}$ → 0.0025 ↳ confuses optimizer s.t. during backpropagation, it still gives some weights to these values, when we shouldn't give at all

$\frac{400}{406}$ → 0.99

$[x_1, x_2, x_3, \ldots, x_6]$

↓ softmax

$\left[\frac{e^{x_1}}{sum}\ \frac{e^{x_2}}{sum}\ \ldots\ \frac{e^{x_6}}{sum}\right]$

↳ $e^{x_1} + e^{x_2} + \ldots + e^{x_6}$ (sum)

→ e^{400^6} → ∞, so basically a value of 1

PyTorch softmax

$\left[\frac{e^{x_1 - max}}{sum}, \frac{e^{x_2 - max}}{sum}, \ldots, \frac{e^{x_6 - max}}{sum}\right]$

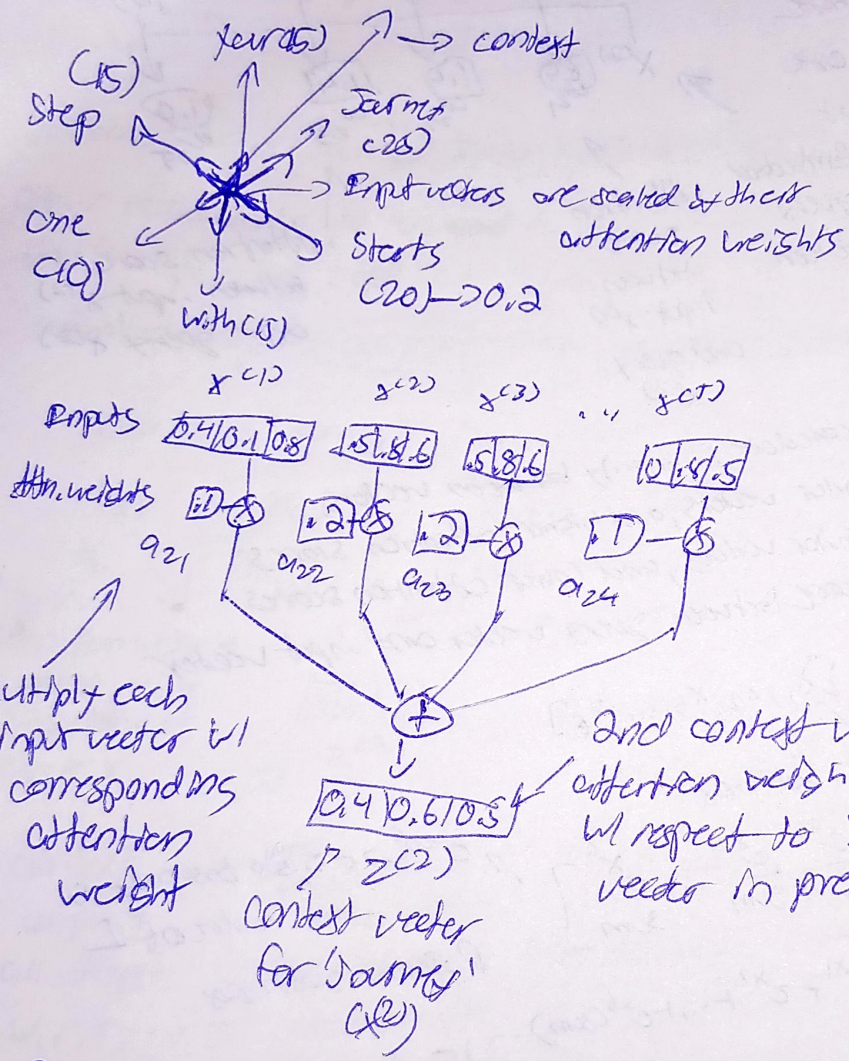
↳ $e^{x_1 - max} + e^{x_2 - max} + \ldots$ →

PyTorch includes −maximum

$\frac{e^{x_1}}{e^{max}}$ → avoid numerical instability from very large or very small values.

Prevent overflow

⑧ After computing normalized attention weights, we calculate the context vector $z^{(2)}$ by multiplying the embedded input tokens $x^{(i)}$, with the corresponding attention weights and then [summing the resultant vectors]



⑨ Compute attn. weights for all input tokens

Now we can extend similar computation as before to calculate attention weights and context vectors for all inputs

1×6 6×6

(10) Follow same 3 steps as before

1) compute attention scores ← dot products between inputs
2) compute attention weights ← normalized version of attention scores
3) compute context vectors ← weighted sum over inputs

Attention weights × Inputs

$$\begin{bmatrix} 0.2098 & 0.2010 & 0.14 & 0.12 & 0.12 & 0.14 \\ 0.138 & 0.237 & 0.233 & 0.124 & 0.108 & 0.158 \\ \vdots \end{bmatrix} \times \begin{bmatrix} 0.43 & 0.15 & 0.89 \\ 0.55 & 0.87 & 0.66 \\ \vdots \end{bmatrix} \begin{matrix} -ir_1 \\ -ir_2 \end{matrix}$$

ac_1 ac_2 — 6×6; w c_1, w c_2, w c_3 — 6×3

$$\begin{bmatrix} \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$ 6×3

$r2c_1$, $r2c_2$, r_3c_3

$= 0.138 \,[0.43\ 0.15\ 0.89]$
$+ 0.233 \,[0.55\ 0.87\ 0.66]$
$\vdots$

~~$r_2ac_1 \cdot ir_1 + r_2$~~

$r_2ac_1 \cdot ir_1 + r_2ac_2 \cdot ir_2 + r_2ac_3 \cdot ir_3 + \dots + r_6ac_6 \cdot ir_6$

0.138 represents "your" attn-weight, so multiply it by "your" vector embeddings and we have essentially scaled it by "your"'s attn. weights, w.r.t. the first word "Journey"