Purpose - The purpose of this memo is explain the intuition, nuts and bolts of the transformer architecture visually.

What is the problem it solves?

- We are given a sequence of inputs $x_1, x_2, x_3 \ldots x_n$ and we want to predict a/sequence of output $y_1, y_2 \ldots y_n$
- This generalized abstraction can be used to solve many problems as illustrated below-
  - given a sentence in english, what would it mean in french
  - given a review of a product, what is the customer sentiment.
  - given an email as input, what should be the response.
  - given an sequence of user interactions with a product (TikTok videos), what are they likely to watch next.
  - given a sequence of words describing a image, what would the image look like.

This abstraction is not limited to the modality of text, and extends to image, audio as well thereby enabling us to solve problems in the field of perception as well.

Why do we need CONTEXT to solve these problems?

- Lets take an example.

As seen in the example above, to understand the meaning of word "He" we need to retain CONTEXT of the last few words, and also pay ATTENTION to the words ELON and LEADER.

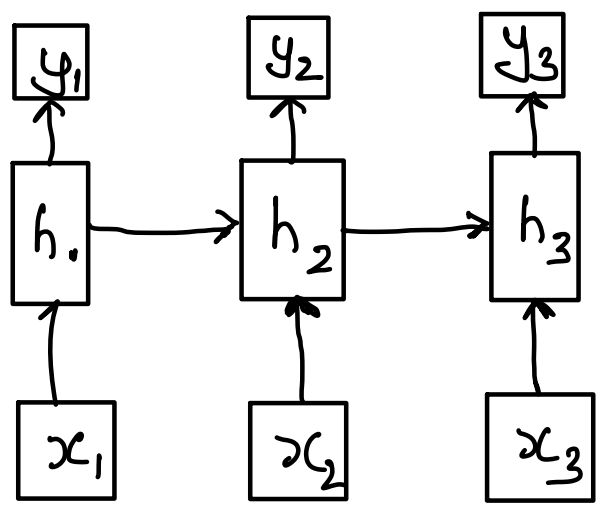
Our brain has memory and we don't understand How but it automatically figures out the right words to pay ATTENTION to find the right meaning.

How do we get computers to do so?

- for computers this can then be modelled as.
  Y = some function $F(X_1, X_2 \ldots X_n)$.
  i.e. to learn some function over a sequence of inputs $X_1, .. X_n$ that can predict Y using the ability to retain. CONTEXT and ATTENTION.

What was the older way to solve this problem?

- have a neural network that can carry forward the past inputs X_1 to X_i when it is working to process X_{i+1} as shown in the figure below-



Insights

- Y_2 uses the influence of x_2 (directly) and x_1 (indirectly) and so on.
- $H_1, .. H_3$ are VECTORS that carry forward the inputs.
- H_i are also called as HIDDEN / LATENT states (for techies)

Let's talk about VECTORS briefly

most quantites cannot be represented with a single number. Naive examples include lat, lng which is 2 Dimensional or velocity which has speed and direction. These representations are called as Vectors.

As an example to find the distance between SFO & LA given their LAT, LNG vectors.

SFO = [lat, lng], LA = [lat, lng].

we can vector difference between the two

$\Delta lat = Lat_{sfo} - lat_{LA}$.

$\Delta lng = Lng_{sfo} - lng_{LA}$.

and then use Δlat, Δlng in the Haversine formula.

In our example from the past "He depends on the word Elon" so it clearly has to be represented using a vector.

ONE-HOT ENCODING.

Elon → [1, 0, 0, 0 .....]

is → [0, 1, 0, 0 ...]

a

great

leader

.

He

makes

revolutionary

things

.

[0, 0, 0, 0 ..... 1]

VECTORS WITH CONTEXT.

[3.0, 2.14, 1.789, ...]

[1.34, 1.92, 2.13, ...]

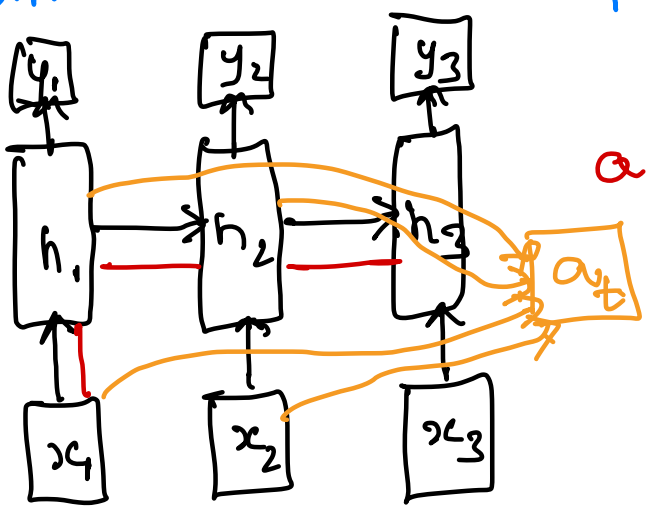
:

[2.51, 18.35, ...]

- In 1-hot encoding, we represent every word in a unique position. It's a simple way but we can't capture any context.
- In the other vector representation, we capture CONTEXT and ATTENTION. This is more rich and useful.
- Now the Problem statement is how to learn these meaningful vector representation over a sequence of DATA in this case text.?

Things like why nueral networks work or the elements that help train them are outside the scope of this memo for now. Lets assume we can train networks on input data, see the error and adjust the weights in the right direction to reduce them. These weights represent a high-dimensional non linear functoon that learns the pattern in the data.

if we had a network that retains memory and context and can attend to different past inputs then we can solve this problem.



- x_1 influence is carried forward through a vector representation **Hi**.
- a_t - is attention. This nothing but weights to each term (past/future) that help us decide how much attention we should give.

This memory based network is called as RNN (Recurrent Nueral Network), LSTM (long short term memory).

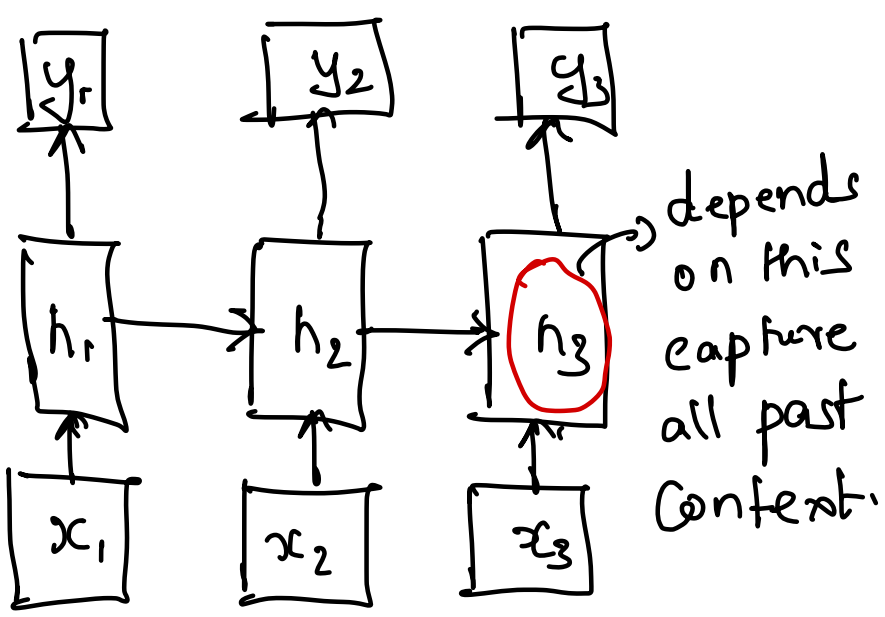
Downside of RNN/LSTM.

- cannot have large context windo (gradient explosion while training - for techies)
- cannot be parallilesed for training as you make the network deeper.

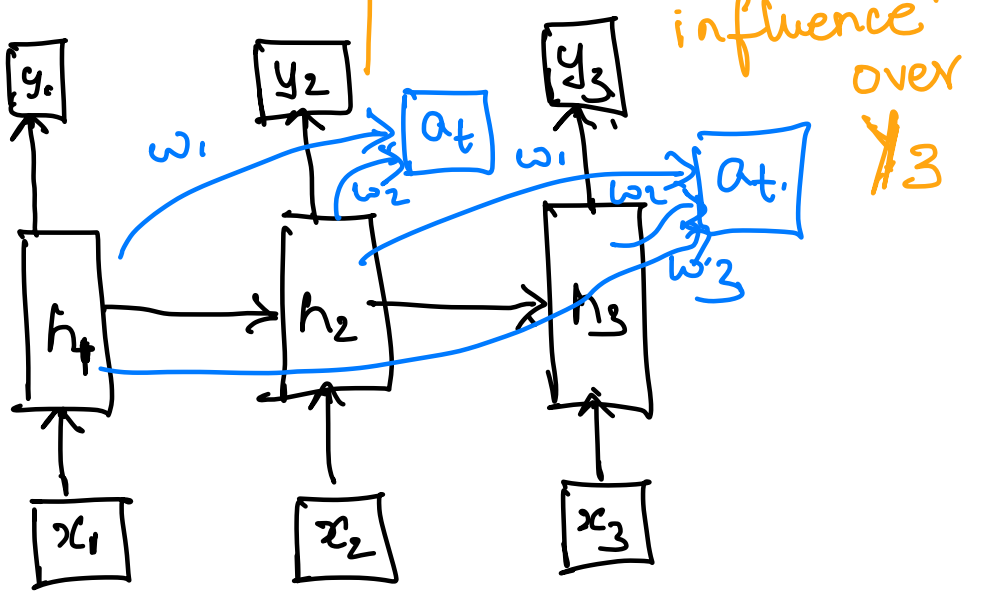
Key insight - The concept of attention was invented before transformers. It is a simple mechanism by which different states (hidden in LSTM) and inputs (tranformer) are weighted and combined together. These weights are also learned as a part of the training process.

Difference between LSTM and attention.

- Instead of depending on all inputs (X_i until now), it attention will consider all. hidden states. This should lead to better CONTEXT.



LSTM.



LSTM with attention.

How do we define the weights in attention?

We learn them depending on the relative importance of the term as a part of the learning process.

We cannot go very far without being able to retain large enough CONTENT. - Why?

Imagine giving the entire book of Shakesphere as input or a thriller novel where the context from the first few & final chapters is needed to figure out the climax. How do we get nueral nehwrks to have such large. CONTEXT.

enter TRANSFORMERS.

KEY INSIGHT- what if instead of paying attention to hidden states, we can directly pay attention to inputs (X_i) then we have solved the problem of trying to maintain a large enough context.

Let's see How.

Why we have chosen 512 as size of vector?
- can it capture all context for X_4. This is move art than science

Let's consider the sequence

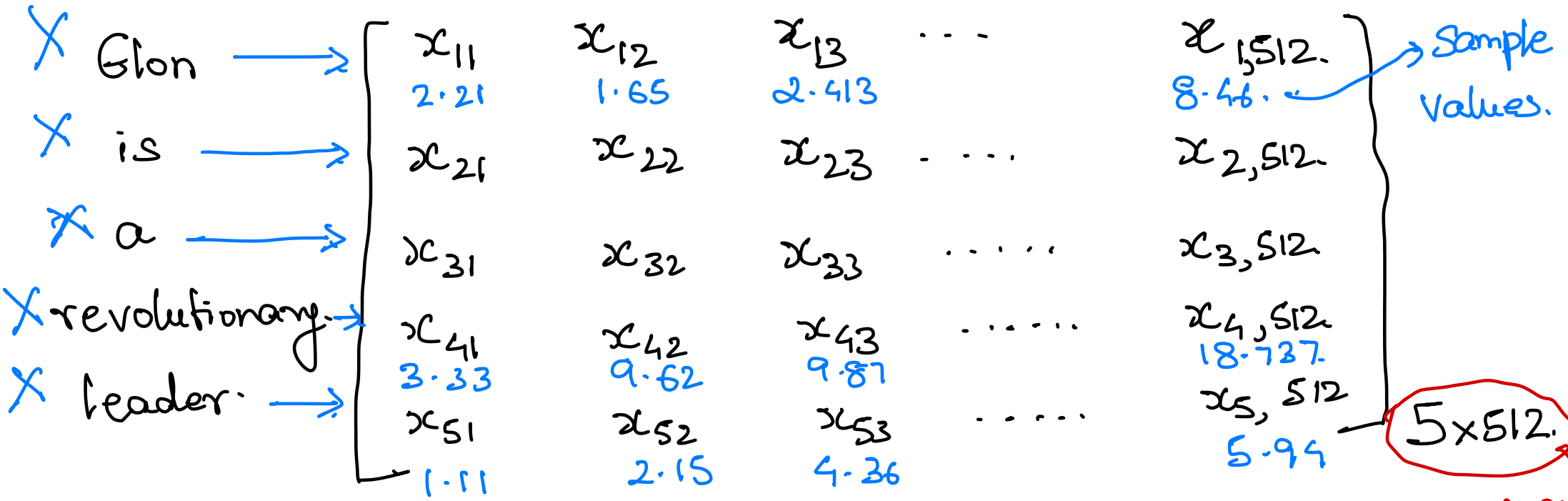
Elon $\longrightarrow X_1 = [x_{11}, x_{12}, x_{13} \ldots x_{1,512}]_{1\times512}$.

is $\longrightarrow X_2 = [x_{21}, x_{22}, x_{23} \ldots x_{2,512}]_{1\times512}$.

revolutionary $\rightarrow X_3 = [x_{31}, x_{32}, x_{33} \ldots x_{4,512}]_{1\times512}$.

leader $\longrightarrow X_4 = [x_{41}, x_{42}, x_{43} \ldots x_{4,512}]_{1\times512}$.

Each of the input vectors $X_1 \ldots X_4$ can be represented as a matrix as shown below. The values of vectors are learned as a part of the learning process.



Input matrix X is also known as

$$X_{Elon} = [x_{11} \quad x_{12} \quad x_{13} \quad \ldots \quad x_{1,512}]$$

and so on.

Vectors are also called as TENSORS. and generally represented as COLUMN MATRICES.

$$X_{Elon} = \begin{bmatrix} x_{11} \\ 2.21 \\ 1.65 \\ 2.413 \\ \vdots \\ 8.46 \end{bmatrix}_{512 \times 1} \Rightarrow X = \begin{bmatrix} X_{1(Elon)}^T \\ X_{2(is)}^T \\ X_{3(a)}^T \\ X_{4(revolutionary)}^T \\ X_{5(leader)}^T \end{bmatrix}_{5 \times 512}$$

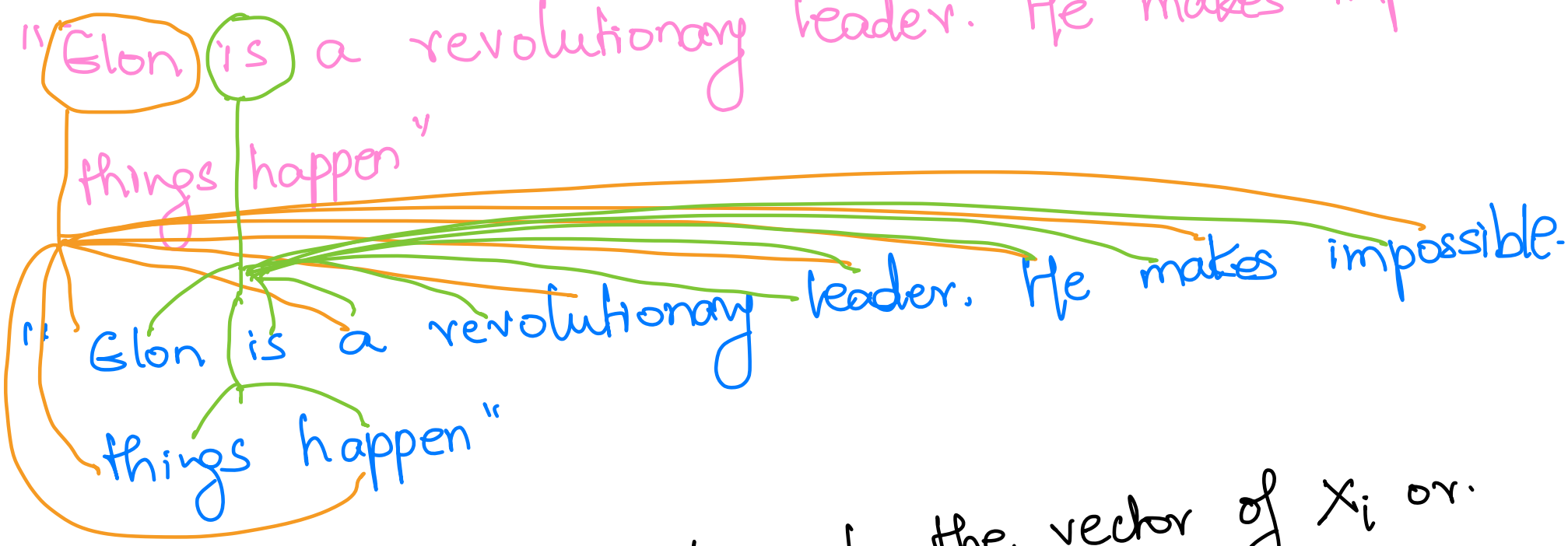
note change in dimension.

Transpose in matrix represented by T, it flips rows to columns & columns to rows.

512x5

each column represents a word here.

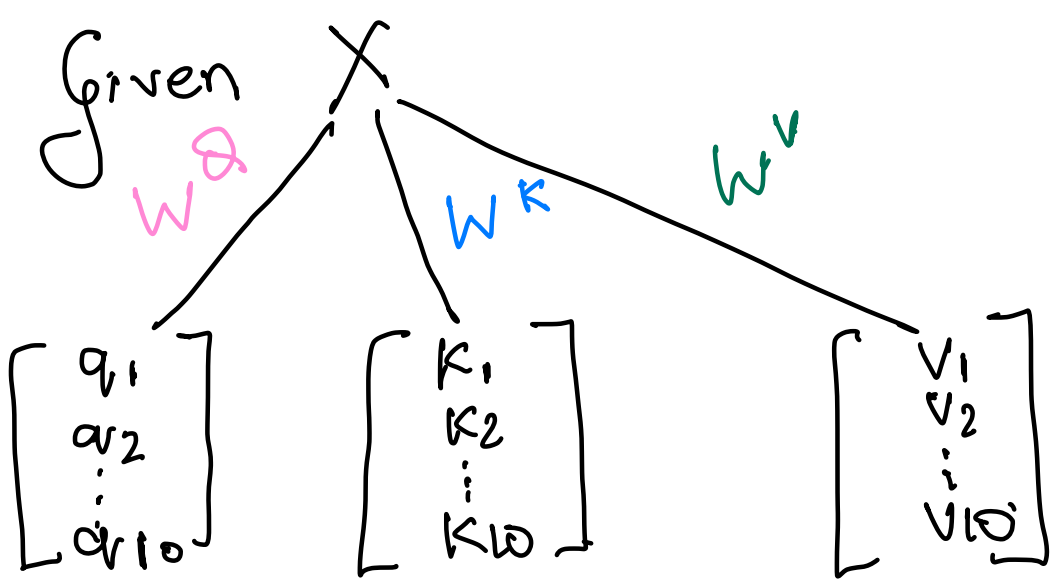
Given these input representations, we have to learn a way that how much ATTENTION we should give to every other word for a given word.



- Mathematically, we have to the vector of X_i or. X_{Elon} and find how similar it is to all the other vectors of all words in X_i. This is visible in the PINK and green arrows shown here.
- Insted of directly working on X_i, we will work on a non-linear representation of X_i. Lets define these as.

Query = Q. (Pink representation of x multiplied by some weights

Key = K. (Blue representation of x multiplied by some weights



- we Know that the best way to learn weights is. a fully connected nueral network.

let's take X_1 from X which the vector for word "Elon"

we know X_1 is 512 rows and 1 column vector = (512×1)

Let's say we want to compress the dimensionalty of the representations of X (Q, K, V) to 64. I think the intution here is from auto-encoder (outside scope) where the network is forced to learn. then.

$$X_1 = (512 \times 1)$$

and $q_1 = (64 \times 1)$

so the weight matrix of q_1 i.e. $W^Q = 64 \times 512$.

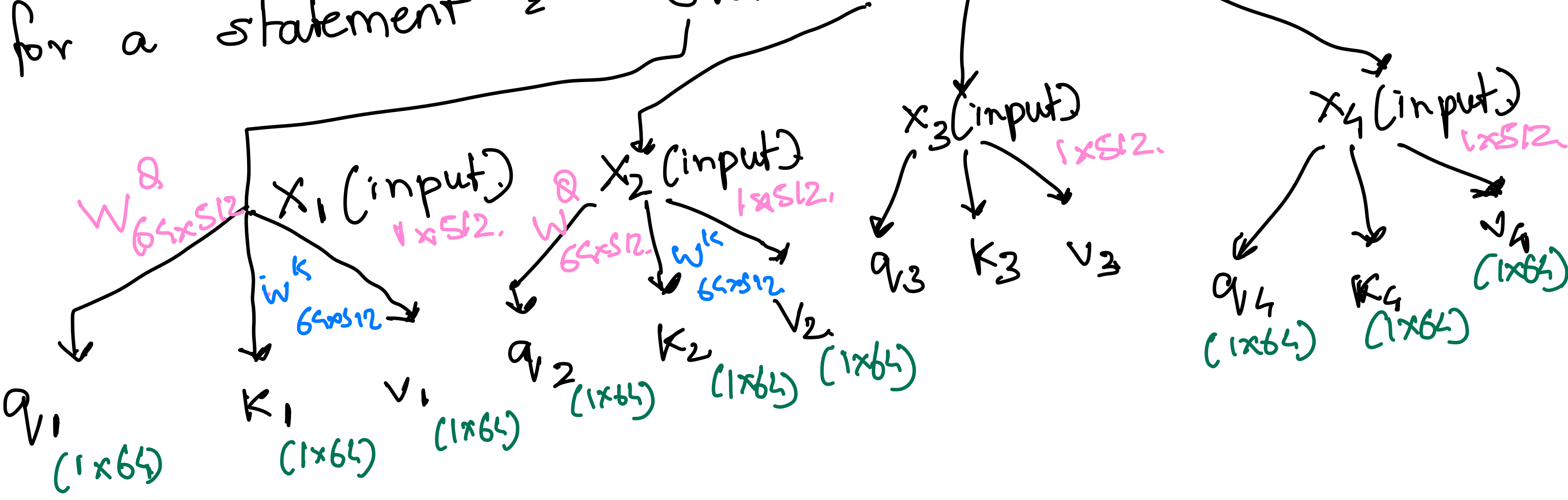
Since $$q_1 (1 \times 64) = \left(W^Q_{(64 \times 512)} \cdot X_{1\,(512 \times 1)} \right)^T$$

Similarly, we will learn a seperate weight matrix to get Key and value representations

They will be defined as.

$$W^K_{(64 \times 512)} \quad \text{and} \quad W^V (64 \times 512)$$

for a statement = "Elon is a leader"



Now, let's calculate "SELF ATTENTION"

$Q \cdot K^T$

query of "Glon"
query of "is"
query of "a"
query of "great"
query of "leader"

$$\begin{bmatrix} q_{1\,1\times64} \\ q_{2\,1\times64} \\ q_{3\,1\times64} \\ q_{4\,1\times64} \\ q_{5\,1\times64} \end{bmatrix}_{5\times64}$$

$$\begin{bmatrix} K_1 & K_2 & K_3 & K_4 & K_5 \end{bmatrix}$$

Key of "Glon"
Key of "is"
Key of "a"
Key of "great"
Key of "leader"

$$\begin{bmatrix} K_{1\,64\times1} & K_{2\,64\times1} & \cdots & K_{5\,64\times1} \end{bmatrix}_{64\times5}$$

Glon
is
a
great
leader.

$$\begin{bmatrix} q_1k_1 & q_1k_2 & q_1k_3 & q_1k_4 & q_1k_5 \\ q_2k_1 & q_2k_2 & q_2k_3 & q_2k_4 & q_2k_5 \\ q_3k_1 & q_3k_2 & q_3k_3 & q_3k_4 & q_3k_5 \\ q_4k_1 & q_4k_2 & q_4k_3 & q_4k_4 & q_4k_5 \\ q_5k_1 & q_5k_2 & q_5k_3 & q_5k_4 & q_5k_5 \end{bmatrix}_{(5\times5)}$$

q_1k_5 dot product represents how close is q_1 to k_5 i.e the word "Glon" depends on the word "leader" in the context of this statement.

This is what we wanted in attention & these weights will give us.

Glon is a great leader.

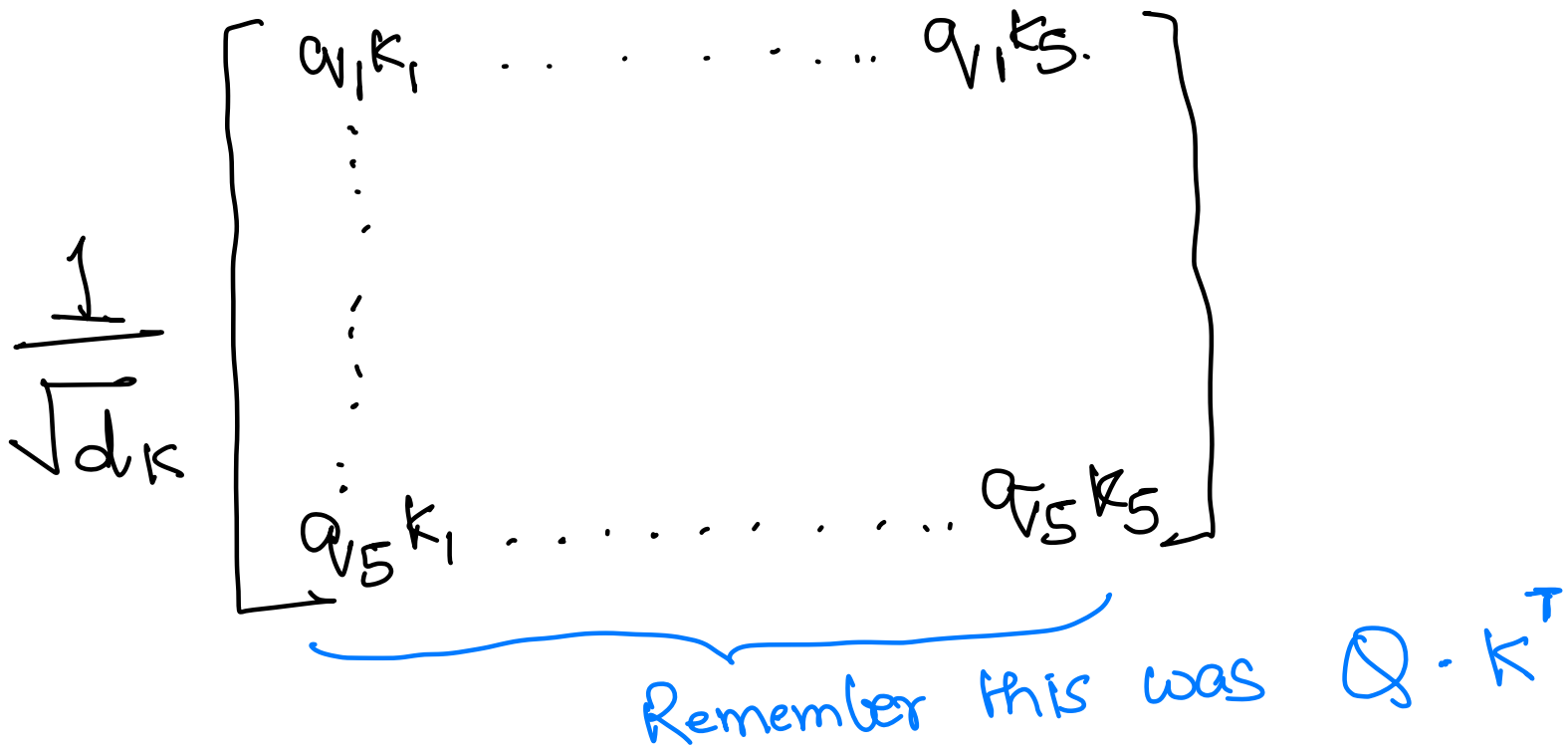
- Using this attention matrix, we can know how much attention we have to give to every other word for a given word.

The size of the attention matrix in this case is 5x5. We took two matrices of size 5x64 and 64x5 to get this.

Now, why do we need to normalize the attention matrix.

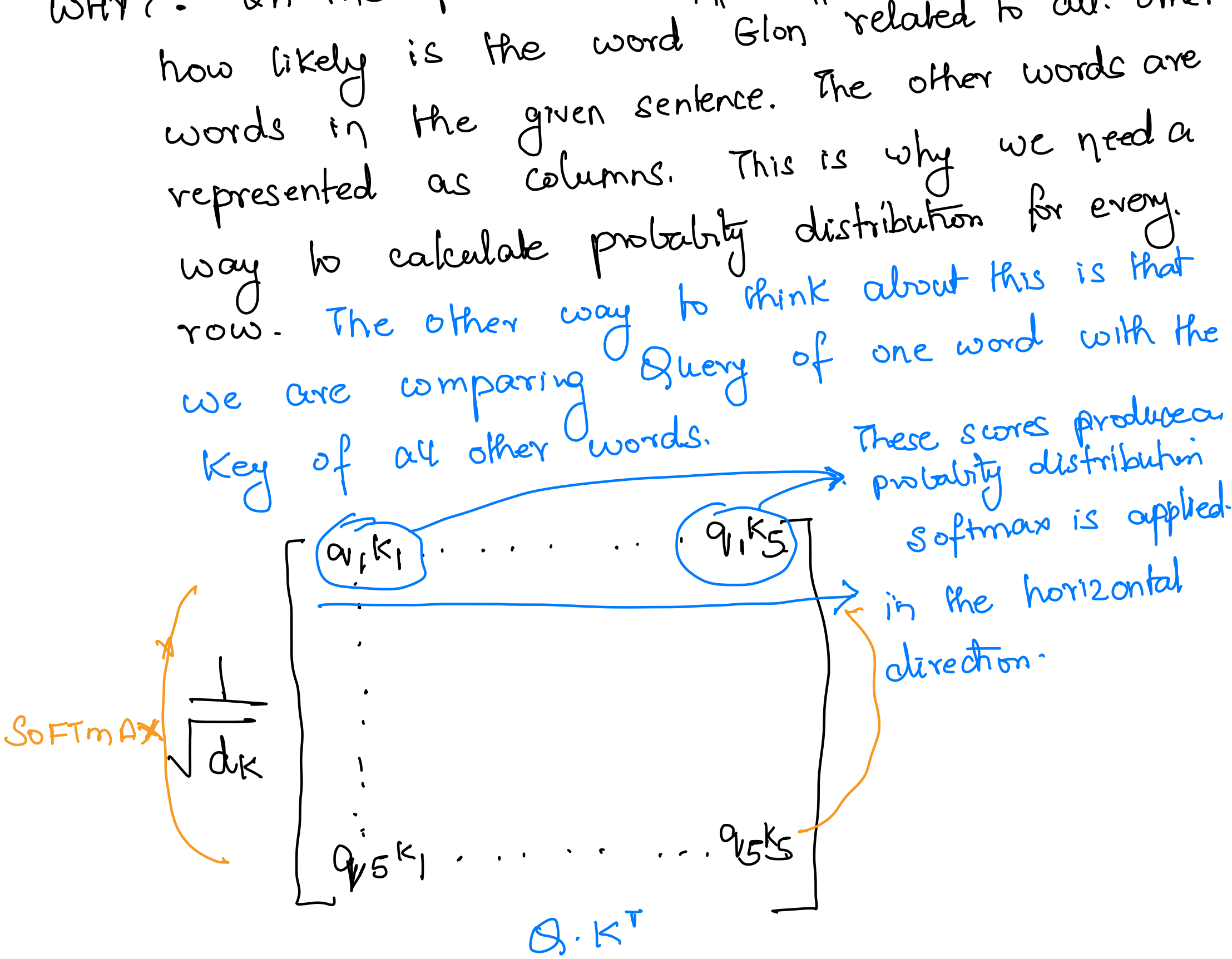
- The dot product between K and Q vectors represent the similarity within them. As the dimensionality of thes vectors (in this example 64) increases, the magnitude of the dot product increases simply because more terms are contributing to the sum. Without normalization, this increase in magnitude can lead to large values in attention scores. This would result in large variance
- Another reason is that after this step we will apply a function called as SOFTMAX to convert these attention scores into probabilites. That function is sensitive to large variations in inputs values, it can lead to gradients that are close to zero, and slow down the learning process. due to vanishing gradient problem.
- The choice of $\sqrt{d_{K\,(key\ vector)}}$ as normalization factor comes from empircal observations. It was found that it prevents softmax function from entering regions of small gradients.

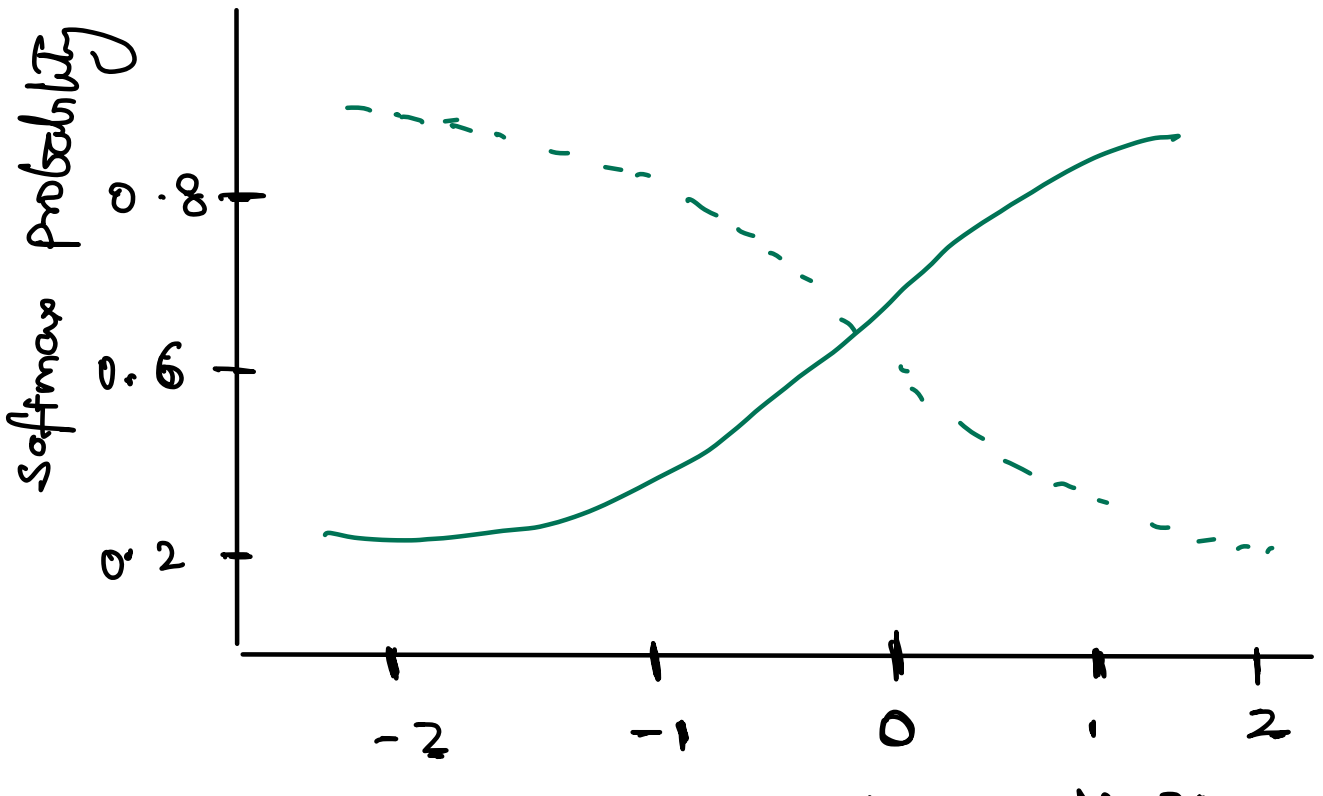
After normalization our attention matrix will look like.



Let's discuss how do we convert these attention scores into probabilities and WHY?

WHY? - In the first row of this matrix, it represents how likely is the word "Glon" related to all. other words in the given sentence. The other words are represented as columns. This is why we need a way to calculate probabilty distribution for every row. The other way to think about this is that we are comparing Query of one word with the Key of all other words.





In this example we vary the input value z_1 keeping z_2 fixed at zero. As you can see as the input value z_1 increases, the probabity of class 1 inoreases and approaches 1, and at the same time probabity of class 2 approaches ZERO.

Softmax takes model outputs and converts them into a probabitty distribution across multiple classes.

The cure shows that even small changes in inputs can lead to significant changes in probabilites especialy when inputs are close to each other. This is a feature of softmax - HOW?

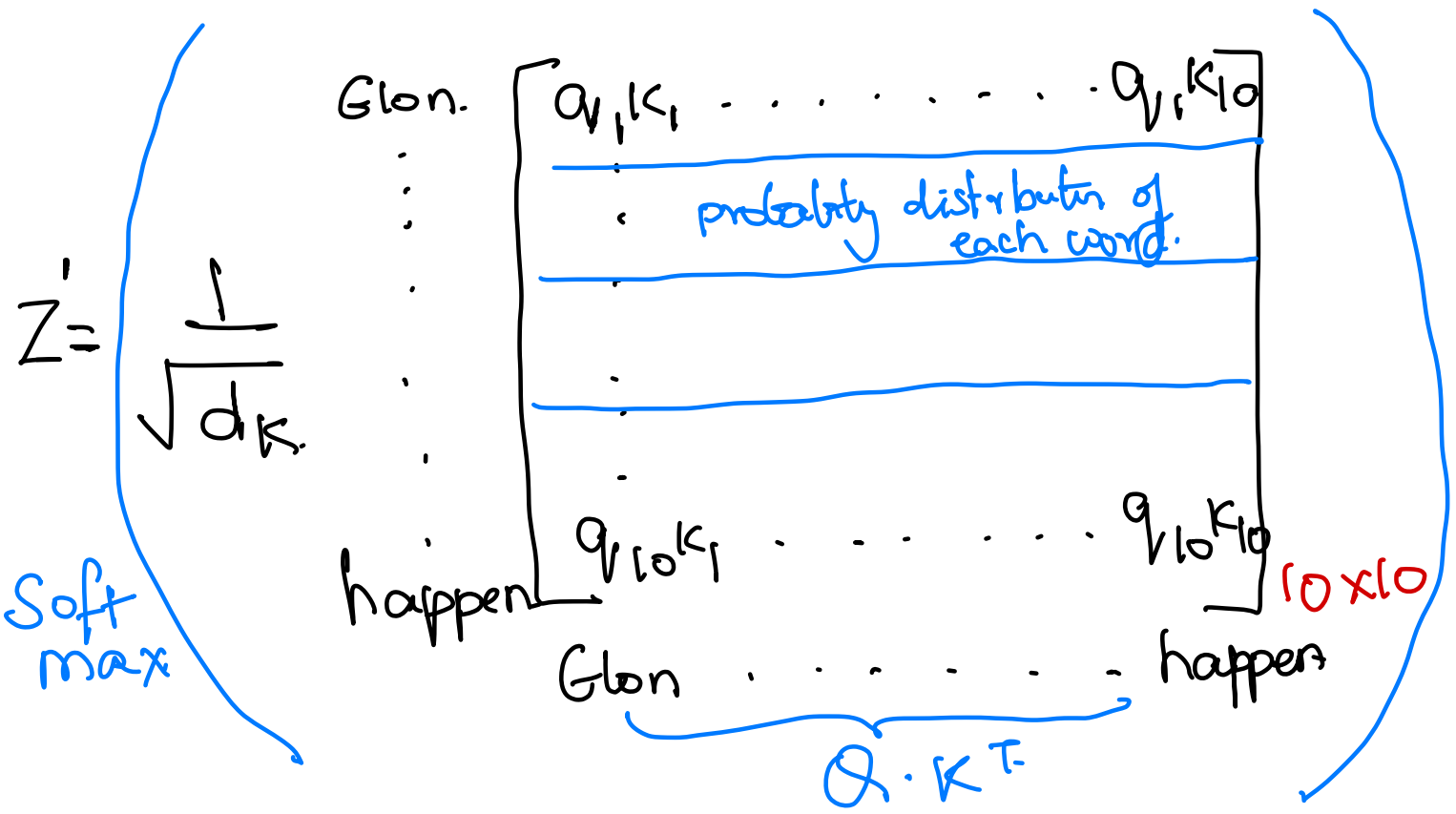
$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

It applies exponential to each value of z_i (VECTOR). This ensures all values are positive and amplifies differences between input values (larger values become significantly larger than smaller values) Benefical for highlighting most likely class in classificatins.

To summarize, softmax does the following.

$$\text{out} = \begin{bmatrix} P(Y = \text{class1} \mid X) \\ P(Y = \text{class2} \mid X) \\ \vdots \\ P(Y = \text{class N} \mid X) \end{bmatrix}$$

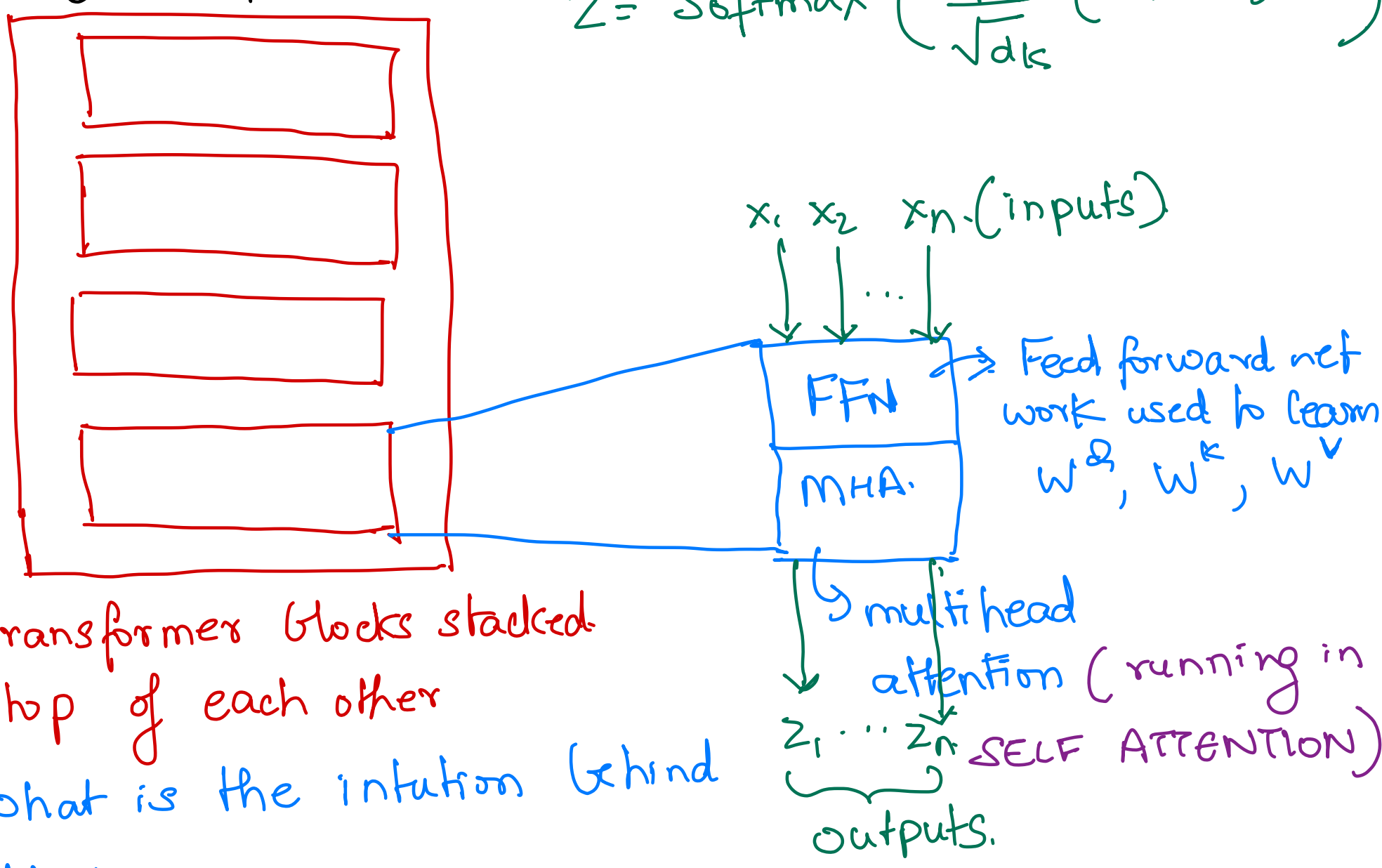
Finally we take the probality distribution of attention scores and multiply it to V (the third non-linear representation of X after Q & K) to apply attention to the given sentence.



$$Z' = \begin{bmatrix} (z'_1) \\ \vdots \\ (z'_{10}) \end{bmatrix} \rightarrow 1 \times 64, \quad \rightarrow 1 \times 64, \quad 10 \times 64$$

Putting it all together -

$$Z = \text{Softmax}\left(\frac{1}{\sqrt{d_k}}(Q \cdot K^T) \cdot V\right)$$



↳ What is the intution behind N blocks

- from my intution of training. CNN's & Deep networks, as you stack up layers, the layers deeper in the network learn higher order abstractions. As an example, if the input is an image, the first layer learns where eyes, nose, ears are in a face. The further down layers in a network use these as features and start to learn how a face looks like (eyes are 6cm apart and so on). With language also, as you read a sentence then the entire paragraph, chapter, chapters, you learn higher order semantics about the language.

- This is visible in the magic of GPT - trained to learn next word prediction but learns all possible things on top

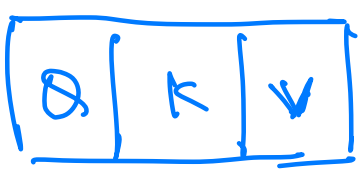
Why do we need MULTIHEAD ATTENTION?

In the given input statement

- "Elon is a revolutionary leader. He makes impossible things happen"

for this word how MULTI-HEAD ATTENTION WORKS.

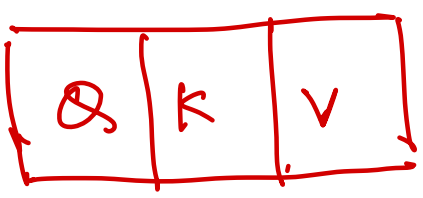
Head1. will have it's own Q, K, V matrices → for a given word/token, let's take "ELON"



and maybe paying attention to some words in the statement. as shown in BLUE.

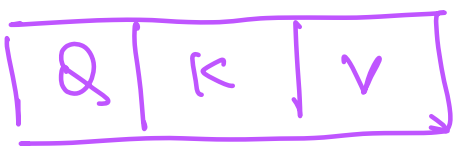
"He" and "Happen"

Head2



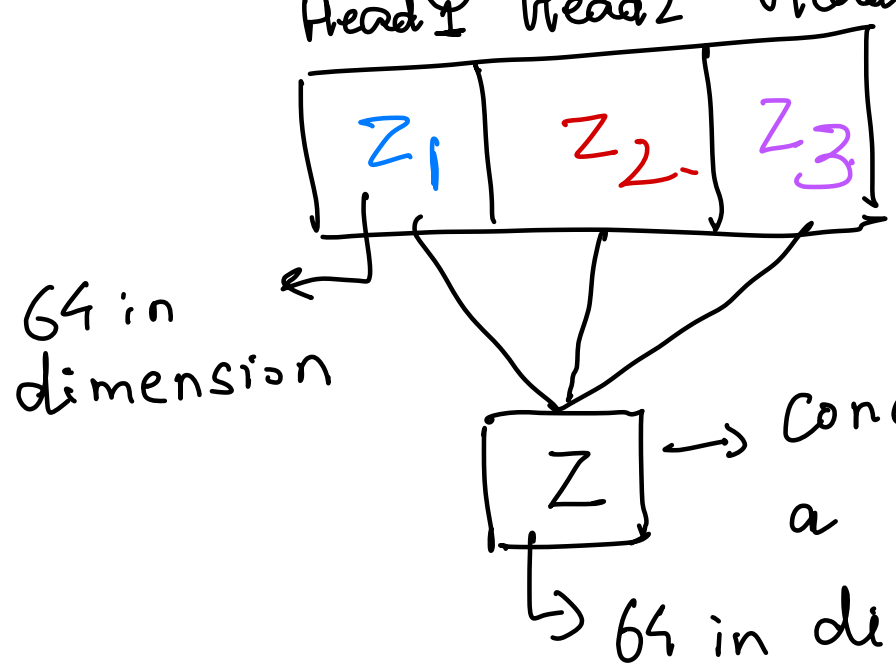
with it's Q, K, V would be paying attention to "revolutionary" and "impossible"

Head3.



with it's Q, K, V would be paying attention to "leader", "He" and "things".

all heads will generate their own output 'Z' matrix.



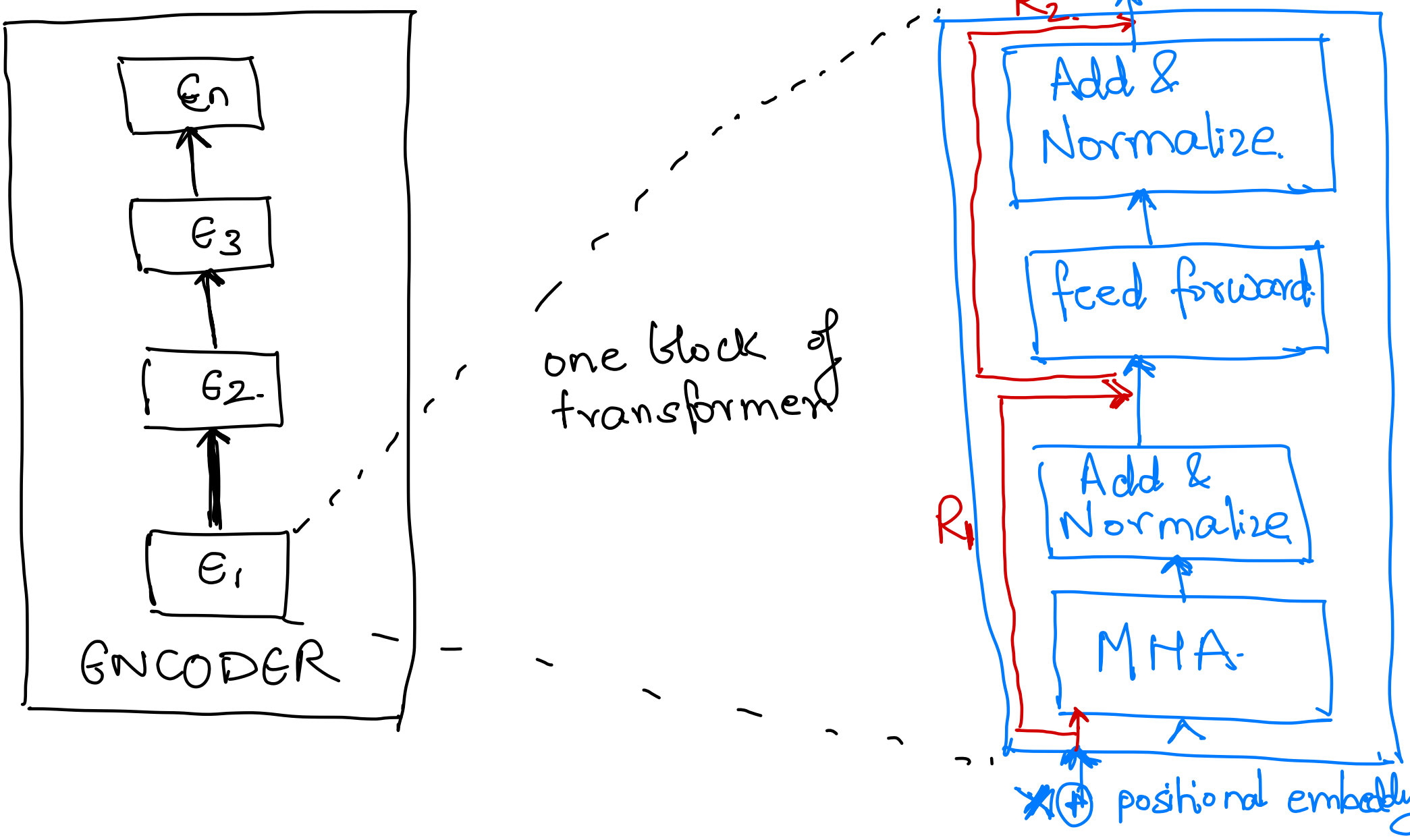
- capture local & global dependencies.

Key INSIGHTS

1) Include various CONTEXTS in training.
2) each head learns it's own content.
3) Multi head leads to a better representation of context.

How are keeping the order of words/tokens in the SEQUNCE.?

- In the current form of encoder if you give the statement "Glon is a revolutionay leader" in any order of words to the ENCODER, the output will be same. We need some way to ensure that the knowledge. that "Glon" is the first word is captured. This will retain the order of sequence and change if we pass a different permutation of input.
- if we change X (input vector) such that we add. $X_i + P_i$ where P is a vector that captures the position of token X_i in the sequence then every. different permutation of X will have a different vector. This is how we taken care of sequencing.



- R1 & R2 are residual connections.

The intution of residual connections comes from RESNET but can be thought of as if deeper layers learn higher order abstractions of images and text then through residual layers we ensure that they still have access to the original input in the deeper layers.

The way they are handled in code is by doing vector addition (element wise) for the vector z_i from layer$_j$ of encoder to x_i & then pass this as input to layer$_j$ of encoder.

- MHA contains self attention blocks (more than one to capture deeper context).

- Normalization used here is layer normalization. Instead of working on batches (like in NN) it works on normatzing every input sample w.r.t. its features. (goto Chat GPT for a better explanation).

What can we build with this transformer based ENCODER.?

- Given text or images as inputs, it will use attention to learn vector representations of these inputs. Using deeper architecture, it will capture higher order abstractions in the inputs

Using these embeddings

1) Build embeddings on amazon review dataset and classify the sentiment of users as Positive or negative.
2) Build embeedings of keywords being used in. Search & products to recommend. Use Nearest neighbour search to find which keywords are more revelant to which product. Build a good enough search experience.
3) Take images of content posted on social media and Build embeedings and then classify. Content as acceptable. or not.

.
.
.
.
.
.

How do we get to the magical state that it starts to generate content like GPT?

- How do we solve the following problems.

a) While typing an email, complete the next few words.

b) Guild something like Chatgpt. - given a prompt of text, complete the next sentence with a response that is useful and meaningful to a fellow HUMAN :)

Let's enter DECODER'S.

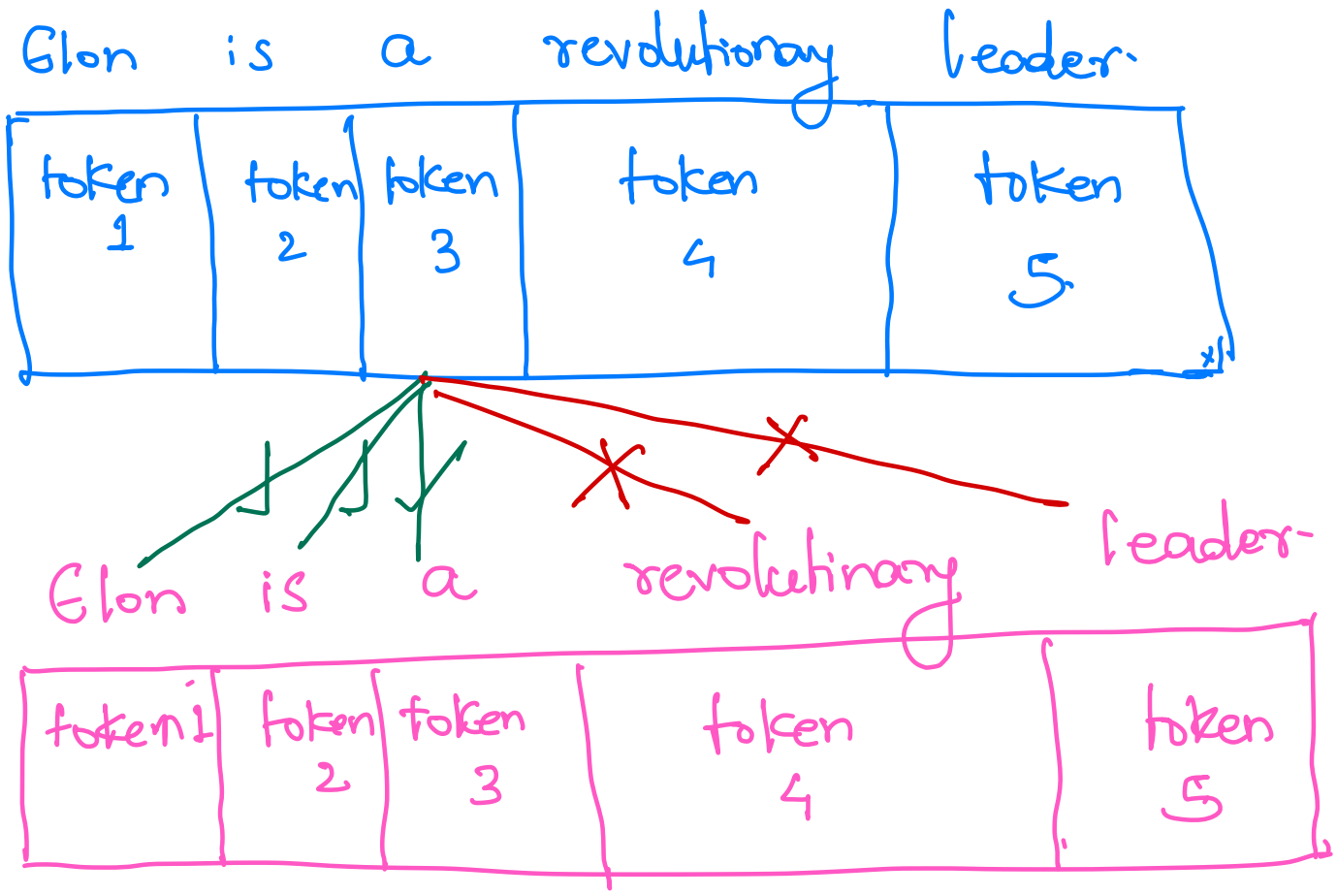
we need an architectural tweak so that the following things get achieved-

1) When we are calculating attention for input X_i, we do not look at any token from X_{i+1} and only look at tokens upto X_i

2) as we generate an output, we feed it back as input to use it as an extended prompt to generate the probability distribution of next word.

Turns out to achieve this, we need just one tweak to our encoder architecture, and specifically the attention blocks.

for input

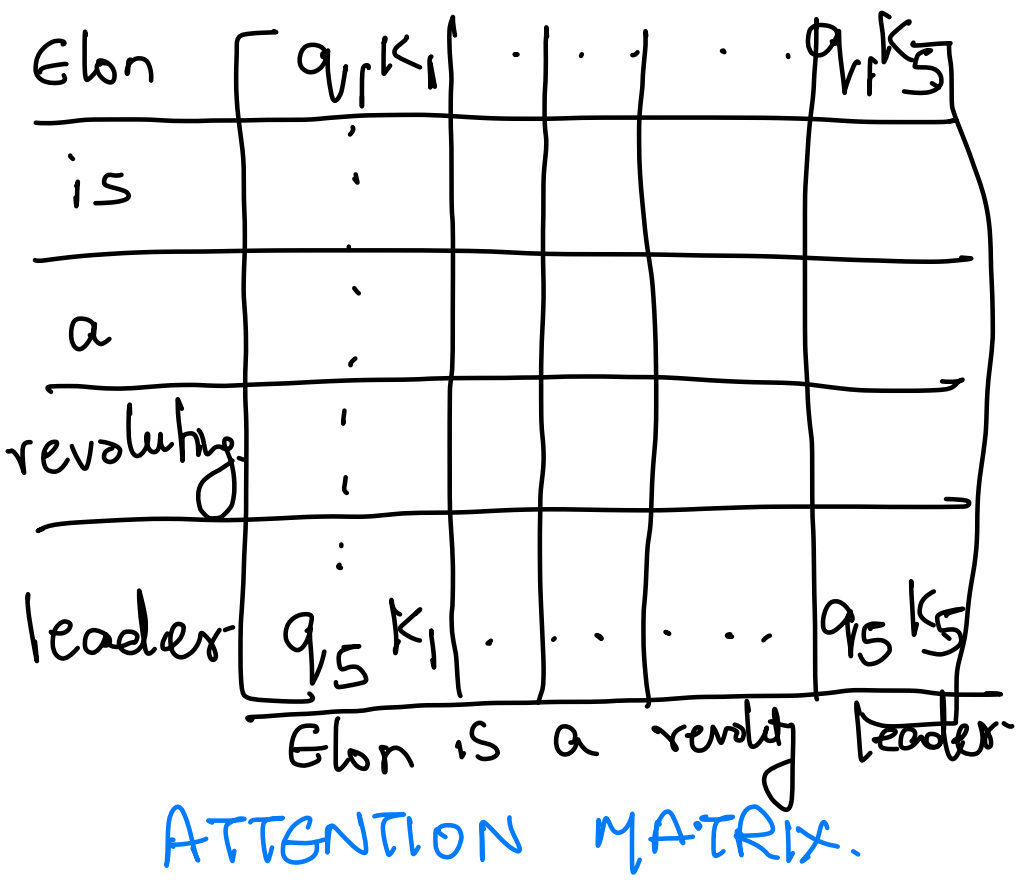


when processing token 3 (a), it will only consider token 1/2/3 for attention and not token 4 & 5.
This allows the decoder to predict next words based on attention scores of previous tokens (1..3).

This is called as MASKED SELF ATTENTION.

we achieve this through the following simple way-

Put negative infinity in positions that don't need attention



ATTENTION MATRIX.

$$-\begin{bmatrix} 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Mask matrix.

$$
\begin{array}{l}
\text{Elon} \\
\text{is} \\
\text{a} \\
\text{revolutionary}
\end{array}
\begin{bmatrix}
q_1 k_1 & -\infty & -\infty & -\infty & -\infty \\
q_2 k_1 & q_2 k_2 & -\infty & -\infty & -\infty \\
q_3 k_1 & q_3 k_2 & q_3 k_3 & -\infty & -\infty \\
q_4 k_1 & q_4 k_2 & q_4 k_3 & q_4 k_4 & -\infty \\
q_5 k_1 & q_5 k_2 & q_5 k_3 & q_5 k_4 & q_5 k_5
\end{bmatrix}
\begin{array}{c}
\\ \\ \\ \\
\text{leader.}
\end{array}
$$

$$
\begin{array}{c}
\text{Elon} \quad \text{is} \quad \text{a} \quad \text{revolutionary}
\end{array}
$$

$\Downarrow$

Softmax to this matrix.

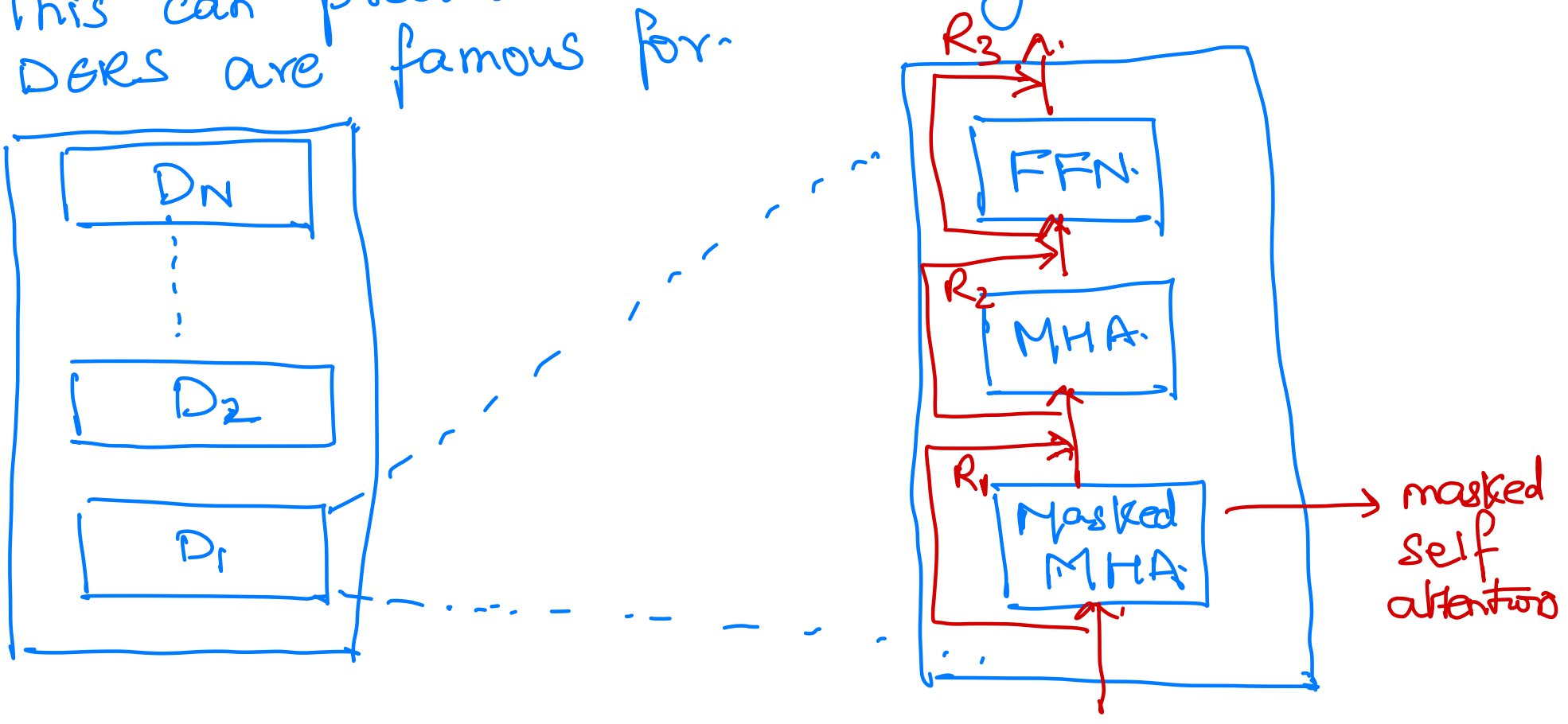
$\Downarrow$

exponent of negative infinity is zero.

$$
\begin{bmatrix}
q_1 k_1 & 0 & 0 & 0 & 0 \\
q_2 k_1 & q_2 k_2 & 0 & 0 & 0 \\
q_3 k_1 & q_3 k_2 & q_3 k_3 & 0 & 0 \\
q_4 k_1 & q_4 k_2 & q_4 k_3 & q_4 k_4 & 0 \\
q_5 k_1 & q_5 k_2 & q_5 k_3 & q_5 k_4 & q_5 k_5
\end{bmatrix}
$$

MSA (masked self attention)

This can produce the auto regresive behavior DECO DERS are famous for.



The magic of ChatGPT?

Turns out if keep training a bigger decoder to predict the next word then it starts to get better at all kinds of language tasks

Bigger => More tokens (dataset)

More blocks of transformers. (deeper)

Bigger context window to calculate. ATTENTION. The size of the matrix.

→ This is the key difference between GPT-2/3/3.5/4 (also has images i.e multimodal).

Training for next word prediction on a bigger context window, dataset and model forces the model to learn higher order representations of data. and become good at all langage tasks - sentiment, summarization, Q&A, paraphrase and much more.

→ They have also applied Reinforcement learning with human feedback on top of GPT answers to learn a policy that will pass answers that humans are likely to accept.

How does language translation work then?

This is achieved through an architectural tweak wherein we put an encoder and decoder together.

English → Hello how are you.

French → Bonjor comment allez-vous

The english text is given to encoder to create the embeddings (latent representation). The decoder uses a mechanism called as CROSS ATTENTION to achieve this. It is depicted below visually.

Key and value matrix come from ENCODER.

Query matrix comes from decoder.

INSIGHT - allows decoder to attends to all positions of input thereby helping it to find relevant ones.