Learning0to1

2025-11-10

The attention step is to study how each word is related to another word. But how would you analyze it? How to analyze the connection between two things? Correlation is one way. It will give you a number telling you how strongly two things are related, between -1 and 1. Is that suitable here?

Remember the goal: to predict the next word. If the leading word is “I”, the next word could be a verb “want, hear, am, think….etc”. Leading word is “ball”, the next word could be “is, game”. So the connection here should focus on how should one word relate to another. But do you want to analyze per word? Or per ‘type’? Instead of focusing on each individual word (what would analysis per word look like? I kind of think the learning process is doing that, but we will dive in later on this). Given a simple word ‘apple’, it could be the object of a verb “eat an apple”, or a noun-noun compound ‘apple pie’, or a special word as Apple Inc. From the other perspective, if the word is ‘eat’, it could be asking for ‘eat apple’, if the word is ‘pie’, it could be asking for ‘apple pie’. So a word’s function has two angles:

1) it could **ask** for a word;
    -- ‘apple’ could ask a noun apple ‘pie’.
    -- ‘apple’ could ask a verb ‘eat’ apple
2) it can also **be asked for.**

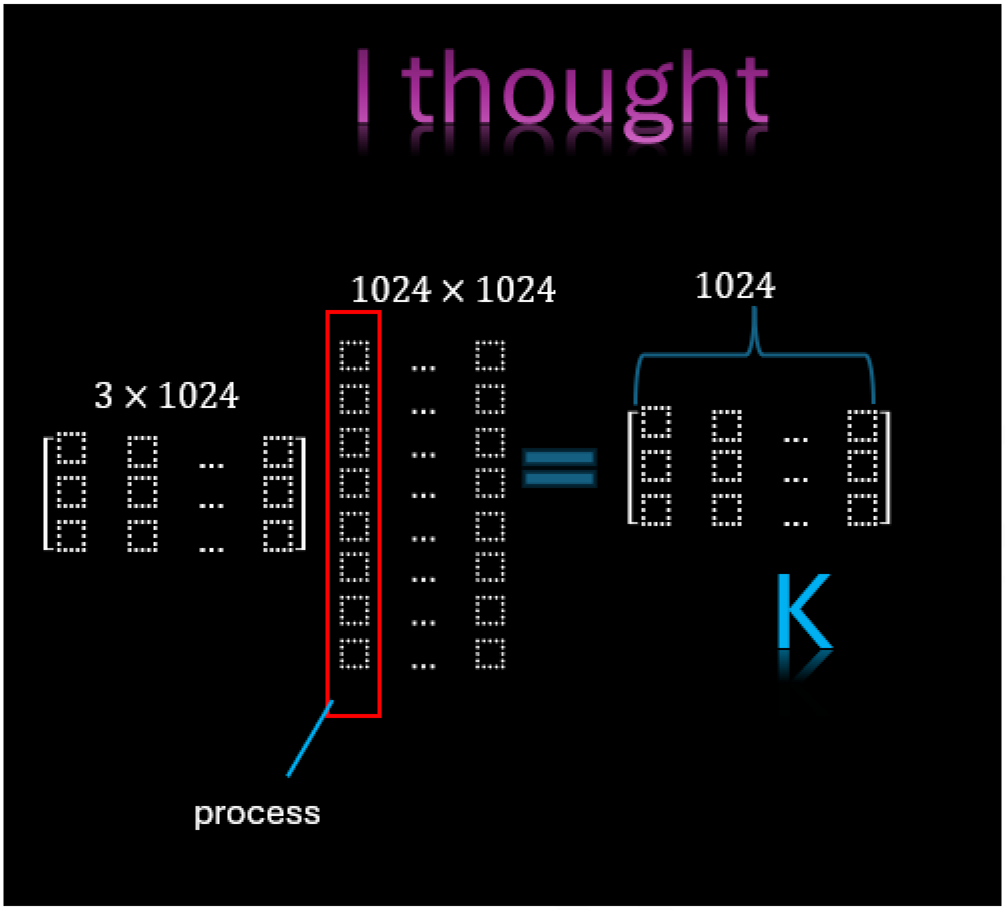
    -- ‘pie’ can ask for ‘apple’ as noun-noun compound

    -- ‘eat’ can ask for ‘apple’

The first aspect is called ‘query’, the second is called ‘key’. Query aspect is easy to understand. Key is easy to understand, but not the name. Ok, so now we understand these two aspects are critical to link two words, now what? Do the analysis, of course! How? Analyze the words of course, ok, so the input tokens 3x1024 (say we have three words each as a vector of 1024 dimension). Then what?? Then linearly mix them. What? Yes, that’s the simplest and most manageable math we can apply to ‘many components’. So by linearly scale them and mix them appropriately, we are analyzing the Query, and Key. But what is the input, what is the output, what is the mechanism?

So in machine learning, to process multiple input simultaneously, linear mixture is the most preferred way. Of course we should ask why, what are the alternatives and why not those? I don’t have the answers right now. Once you accept that, that is, the ‘process’ will always be through a matrix, matrix multiplication is the mechanism.

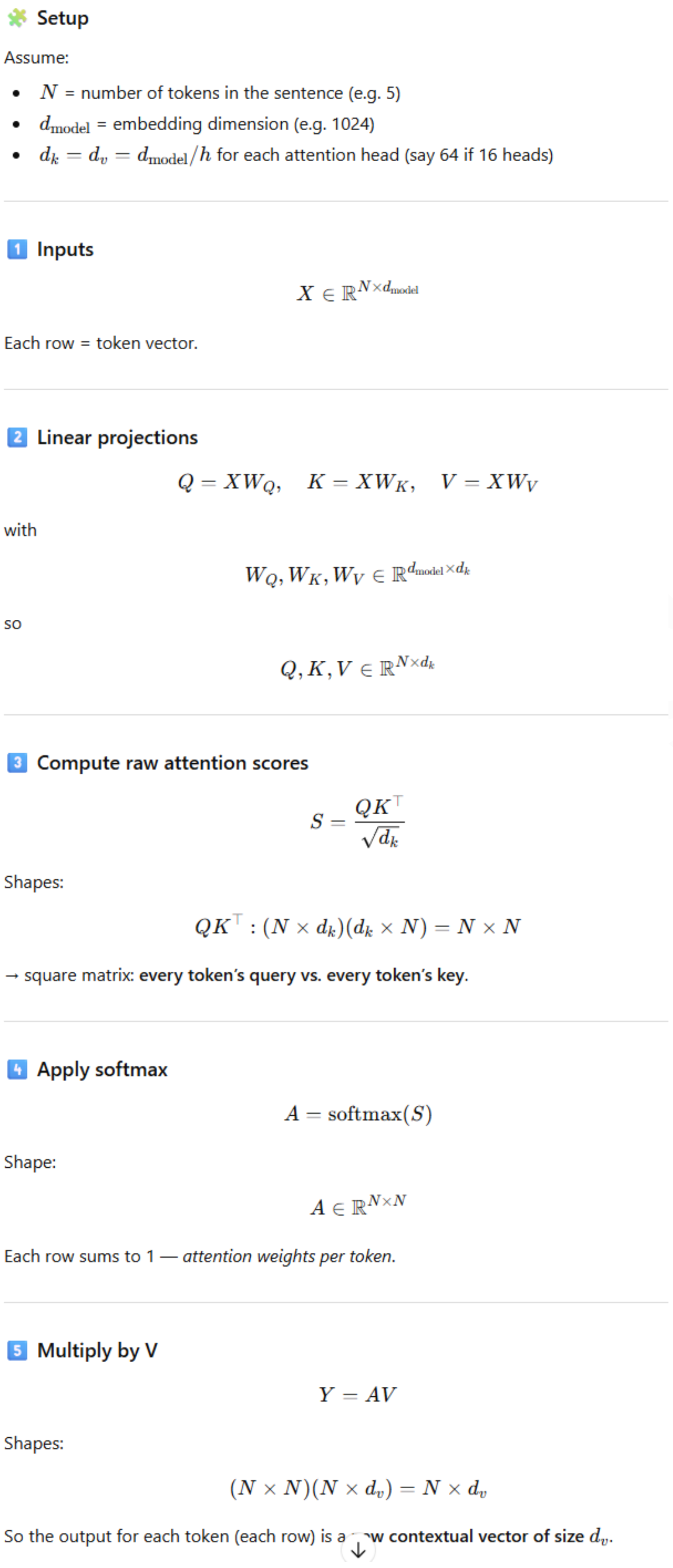
This is what I thought:



The W_K matrix will be 1024x1024, because I was thinking the output K should still be in meaning space, thus retaining the 1024 dimensions. But GPT told me, no, the output will no longer to be in meaning space, and he called that 'feature' space?

Let's follow what GPT is saying.

Learning0to1



🧩 Setup

Assume:

- N = number of tokens in the sentence (e.g. 5)
- d_{model} = embedding dimension (e.g. 1024)
- $d_k = d_v = d_{\text{model}}/h$ for each attention head (say 64 if 16 heads)

1️⃣ Inputs

$$X \in \mathbb{R}^{N \times d_{\text{model}}}$$

Each row = token vector.

2️⃣ Linear projections

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

with

$$W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$$

so

$$Q, K, V \in \mathbb{R}^{N \times d_k}$$

3️⃣ Compute raw attention scores

$$S = \frac{QK^\top}{\sqrt{d_k}}$$

Shapes:

$$QK^\top : (N \times d_k)(d_k \times N) = N \times N$$

→ square matrix: **every token's query vs. every token's key.**

4️⃣ Apply softmax

$$A = \text{softmax}(S)$$

Shape:

$$A \in \mathbb{R}^{N \times N}$$

Each row sums to 1 — *attention weights per token.*

5️⃣ Multiply by V

$$Y = AV$$

Shapes:

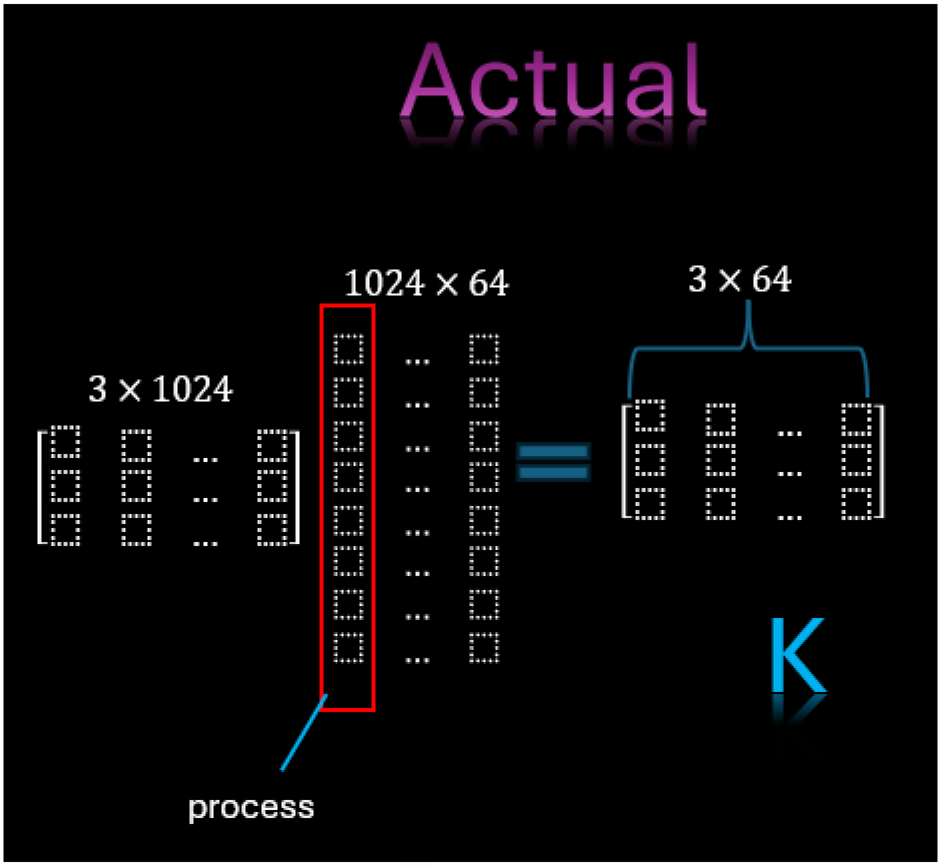
$$(N \times N)(N \times d_v) = N \times d_v$$

So the output for each token (each row) is a [illegible]**w contextual vector of size d_v**.

So say the word has a dimension of 1024, we partition into 16 heads, so each W matrix is going to be 1024x64. After the token-W multiplication we get what? We get Query, Key and Value:

$Q = XW_Q, K = XW_K, V = XW_V$.

Each one is 1024x64.



So by linearly mix and combine the 1024 meanings, we create keys. We know what Key mean, but how it is represented? Again a 64 dimension vector. In other words, each key is not a number, but a list of numbers. Why? Why scalar is not enough? Also why 64 dimensions? You can think of the 64 columns of W_K as 64 different processes to mix and combine meaning components differently in order to create 64 key-components.

Then separate W matrix will create Q, and V, each of 3x64. So each row of Key is a key for a token, each row of Q is a query for a token, each row of V is a value for a token.

Then compute how good the match is between Q and K: $S = \frac{QK^T}{\sqrt{64}}$. This step is to compute how similar/matching Query is with Key. So you need to understand dot product of two vectors measures the angle between them, hence the difference/similarity between two things. (any other measures? For scalars, difference, log ratio would do the work, but in this case we are measure how two things are different, angle is the best way.). I have two questions:

1. Is Q and K in the same space? I would think so.

2. How to interpret the diagonal components, S_{11}, S_{22}, S_{33}? How each token can be matched by itself?

But we have 16 heads, so each head will give us a similarity score. So GPT says we would concatenate 16 3x64 K, Q, V matrices respectively, so each K, Q, V will be 3x1024. Ok I am not understanding. Ok, asking GPT, he says we will have 16 such similarity matrices. So similarity from 16 perspectives then. Ok. Since S is full of cosine, it should be between [-1,1]. Now we know how one word matches the other, in terms of cosine, then what? Then we calculate the weight, called 'attention'. But why not straight weight?

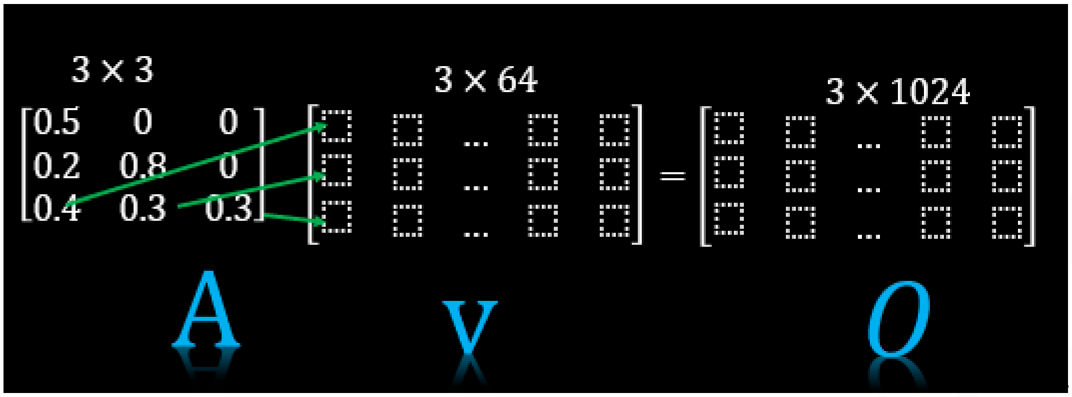
Attention is between words. $\begin{bmatrix} S_1 1 & S_1 2 & S_1 3 \\ S_2 1 & S_2 2 & S_2 3 \\ S_3 1 & S_3 2 & S_3 3 \end{bmatrix}$. I used to find it difficult to explain the meaning of diagonal terms, because I tend to explain these elements as interactions between one and another, sounds reasonable until you meet diagonal: what do you mean my interaction with myself? But in this case, totally explainable, because each entity (token) has two aspects: its need, and the need it can satisfy. Each number measures how one can satisfy one's need. S_{11} measures how the first token can satisfy its own need (query), similar to S_{22}, S_{33}. S_{23} measures how token 3 can satisfy token 2's need.

After quantifying how one word matches the other, then calculate the 'attention' between them. The logic is, if you are a good match to me, I should pay more attention to you.? Or, what is that? Or can we just say : I will put more weight on you? But then it sounds strange: why would a single word need to put weights on other words? I think it is because of the special 'mechanism' used to 'aggregate a few words meaning': a sweeping manner.

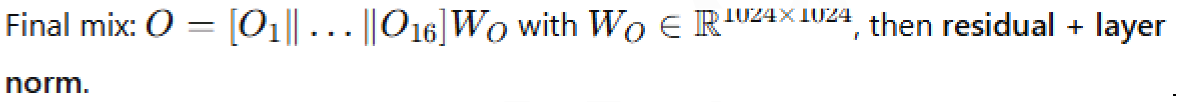
Word1 word2 word 3 word 4 word5 word 6 word 7 word 8 word9 word 10. See the 'contextual meaning' depends on all the words, hence mathematically you just add words' meaning together now words are numbers. But with different weights.

Say our sliding window is of 10, so combine 10 words together. It's done in a sweeping manner I think. First word meaning as $A_{11}t_1$ (?), now scanning the second word, it is going to be combing its own meaning with the first word, so $w_1t_1 + w_2t_2$, but how much weight w1 and w2 are? I think that's the attention coefficient? $A_{21}t_1 + A_{22}t_2$. Now scanning the third word, $A_{31}t_1 + A_{32}t_2 + A_{33}t_3$. Hmm, something is not right! What happens to the previously combined word $A_{11}t_1$ and $A_{21}t_1 + A_{22}t_2$? Aren't they supposed to be swept in? but anyhow, this is a detailed question we will ask GPT later to clarify exactly how are those attention (weights) used. Ok, say we are good with the general idea of calculating attention matrix as weights, we understood the cosine part, now what is softmax, a little googling shows it is a normalization scheme using exponential function. So nothing fancy there. We can pass for now.

Now each attention matrix is 3x3, evaluating how good match one token is to another. Now what to do with them? We multiply them by the value vectors. Ok, so we did not really explain the meaning of 'value', since we focused too much on "query", and "key". Query focuses on what a word might need, key focuses what a word can provide (as a noun-noun compound, as a special iconic token such as "Apple", as an adjective modifier, etc). These two provide the means for us to check if two words are good match or not. But then what? Say they are good match then do what? That's what "Value" comes from, they represent what a word provide as value. See I used to be so confused about this, because then it sounds the same as "key", but key focuses on function, like, a noun-noun compound, instead value focuses on 'what they are' (need better explanation!!). Say 'pie' is a good match for 'apple' as a noun-noun compound, but ~~then once we match apple to pie, then do what?~~ I crossD this out because we only check if they match, when did we ever match them? So I believe AV is doing that.



If you look at the above mechanics, AV operation the make sure each word's meaning will be multiplied by its weight (attention) accordingly. For example A_{21} is saying the first token only matters to the second one 20%, while the second itself weighs 80%, hence to mix the meaning, only take 20% of the first word, 80% from itself. So 16 different ways to match gives us 16 results. Then just append them all together???? Are they in the meaning space? The dimensions match, but are they meanings? Don't think so! Look at the final step GPT has to do. It needs to do a linear transformation once more!


Final mix: $O = [O_1 \| \ldots \| O_{16}] W_O$ with $W_O \in \mathbb{R}^{1024\times1024}$, then **residual + layer norm**.

Though I am not sure the meaning of 'residual+layer norm', we will ask later.

Learning0to1

2025-11-11

So let's recap. The goal of this attention step is to create 'contextual meaning' from individual words. But remember the 'context' is based on each word: $token_1$ $token_2$ $token_3$ $token_4, token_5$ …. In a nutshell, we just need to know the weight of each token and then weight average them: $w_1 t_1 + w_2 t_2 + w_3 t_3 + \cdots$.

But how to get the weights? How to decide the 10%, 20%, 15%, weights? And a question prior to that is: what is the group size you are looking at? Remember when we human read a paragraph, we can tell the 'chunks' of the message, ie, we only look at one sentence at a time, because they are meant to convey separate/independent meaning. Sometimes 3 words is a sentence, sometimes 10 words, if too long, we will break into several parts to assess. So the "weights" seem to have a shifting group size. But how would robots distinguish that? Sure, punctuation symbols are good way, but no guarantee since the writing might not always be in the standard way. It almost seems that we are in a chicken-egg loop:

1) To get the contextual meaning, we have to assign proper weights to each word
2) To get the proper weights, we have to at first know the group size of the words. But that almost requires us to have some primitive contextual meaning to start with, in order to know when to consider 4 words as a group vs 10 words as a group.

Sounds like a non-starter?

So I would consider the way weights are handled in LLM as a ~~breakthrough~~ milestone, I mean, not many people would know this is the way to do it if they were not told so. It considers the weight on a rolling basis, from a token's perspective. So the first token only cares about itself, the second one cares about the first one and itself, the third one cares about the first, second and itself. In other words, any token will take a look at the previous tokens up to itself.

" it is not a cat". When we assess the importance of the words, we seem not to have a baseline to compare with. We would think 'not cat' are the keyword here, so they should have more weights. Seems that we are evaluating a word's importance over its overall meaning already.

But LLM handles it differently. It assess a word's importance to one another! Then attention matrix is the math mechanism that does so. Each word will assess how important the previous words are to itself.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

Each row i is for word i, it is assessing how important other words are to it. For example, the first row is for word1, A_{12} is the importance of word2 to word1, A_{13} is the importance of word 3 to word1. It is easier to use this little arrow pointing from the column index to the row index, reminding us that it is from the row's perspective. But what about A_{11}? It's weird to talk about how word1 can be important to itself. We will get clearer on this later.

$$A = \begin{bmatrix} A_{1\leftarrow 1} & A_{1\leftarrow 2} & A_{1\leftarrow 3} \\ A_{2\leftarrow 1} & A_{2\leftarrow 2} & A_{2\leftarrow 3} \\ A_{3\leftarrow 1} & A_{3\leftarrow 2} & A_{3\leftarrow 3} \end{bmatrix}$$

Learning0to1

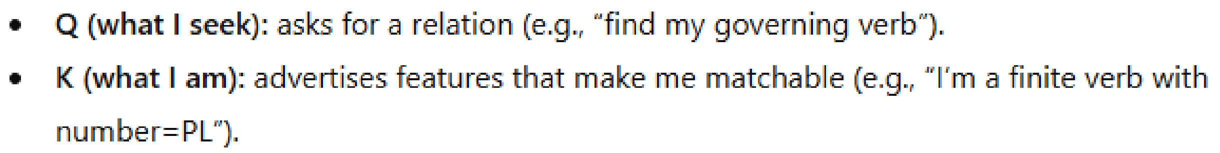
But remember LLM considers the words in sequence: only previous words will impact later words, not otherwise, hence the first word only matters to the later words, while the later words do not impact it. Hence, this matrix will need to be an upper diagonal. (I read this part somewhere else I think, I haven't asked GPT yet).

$$A = \begin{bmatrix} A_{1\leftarrow 1} & 0 & 0 \\ A_{2\leftarrow 1} & A_{2\leftarrow 2} & 0 \\ A_{3\leftarrow 1} & A_{3\leftarrow 2} & A_{3\leftarrow 3} \end{bmatrix}$$

So LLM analyzes words' importance on a pairwise, rolling basis. They call that importance 'attention', the attention one word should pay to another. Hence A_{31} is how important word1 is to word3; A_{32} is how important word2 is to word3.

Once you understand this piece, the next question is: how to assess how important one word is to another? (But I see another question: is this enough? The pairwise, rolling approach? We will deal with it later). Now we will talk about how to assess the importance of one word to another.

Now you need to know the double role of a word: it could be wanted, it could also want a word. Like seeing a verb 'kick', you might wonder: who kicks, kicks what? Seeing 'football', you might wonder what happens to this football? Is it kicked? Is it damaged? Is it rolling? Or what? While 'kick' might need an object like 'football', 'football' might need a verb like 'kick'. These dual aspects have special names: Query and Key. Query is about a token's need, Key is about what a token provides. Here is what GPT says:



- **Q (what I seek):** asks for a relation (e.g., "find my governing verb").
- **K (what I am):** advertises features that make me matchable (e.g., "I'm a finite verb with number=PL").

From the perspective of 'football', its query could be asking for 'finding my governing verb', at the same time it is a noun which can be acted on.

So the next step is to study each token from these two perspectives: Query and Key. But how? The answer is a linear model. To analyze the words, linearly combine the meaning components. To analyze words' Query, linearly combine the meaning components, to analyze words' Key, linearly combine the meaning components. In this LLM network, anytime doing any analysis on a vector, just linearly combine them! So the question is: that's it? Is that enough? The model reads the same, how can you be analyzing different things? I think that's controlled by the different parameters. I am not expecting you understand this yet, because to be honest, I still find it hard to believe: linear model is good enough? To gain better understanding about this, maybe we should briefly review neural network. I only took NG's simplest machine learning course, where you combine layers of linear model plus non-linear activation function, hoping that nested structure can mimic some complex relationship you are trying to model. So my understanding about neural network is, it gives us, human, a manageable method to treat real life complexity. I will do the digression later, now let's stay with attention model.

The different parameters are placed in a W matrix, W_K for analyze Key, W_Q for analyze Query. The analysis, or the linear combination process can be represented by matrix-matrix multiplication with X storing words row wise: $\mathbf{K} = \mathbf{XW_K}$, $\mathbf{Q} = \mathbf{XW_Q}$. What about the dimensions? But is that a good question? Sure, for anyone taking linear algebra knows it's time to check dimension: $X\ is\ (3 \times 1024)$, if we are talking about 3 words; $W_K\ is\ (1024 \times ?)$. How many columns should W_K have? Each column means 'process', so how many different analysis we should do on words? I don't think there is a fixed number yet. GPT says 64 for now. So the output will be 3x64, each row is for one token, then what does 64 dimension mean? Maybe we should ask: what does the output K mean? No doubt, it represents the Key for each token. But how come it has 64 elements? Well, I can only tell you that, that means Key is not a scalar either, it also has many dimensions. So a word is a vector of 1024 (or more) dimensions, its Key/Query are also vectors of multiple dimensions. Why? This question does not have simple answer: so maybe we should ask, why simple scalar is not enough? We will drill down later on this. Now attention matrix.

After we get the Key and Query matrix, each 3x1024, we can check if they match! The dot product between two vectors measures their angle, hence their similarity. So the QK^T step, now is checking how each token's key match another token's query!

Learning0to1

2025-11-12

In short, separate processing matrices W_Q, W_K turns input tokens into Query and Key:

$$\begin{bmatrix} X_{11} & X_{12} & \dots & X_{1,1024} \\ X_{21} & X_{22} & \dots & X_{2,1024} \\ X_{31} & X_{32} & \dots & X_{3,1024} \end{bmatrix} \begin{bmatrix} W_{11} & W_{12} & \dots \\ W_{21} & W_{22} & \dots \\ \dots & \dots & \\ W_{1024,1} & W_{1024,2} & \dots \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} & \dots & Q_{1n} \\ Q_{21} & Q_{22} & \dots & Q_{2n} \\ Q_{31} & Q_{32} & \dots & Q_{3n} \end{bmatrix}$$

I tend to interpret matrix as representing the dynamics/interactions within a complex system, in a pairwise manner. But in the case we stack tokens together to create an input matrix X, I don't see any such significant meaning, right now merely looks like 'bookkeeping'. So is the output. But anyway, the processing matrix 'analyze each token' and produce their "Key" and "Query" matrices. The Key and Query matrices have in them each row representing the corresponding Key and Query for a token. For example, the second row of the query matrix represents the Query for the 2nd token, the second row of the key matrix represents the Key for the 2nd token.

Now the key matrix has the same structure,

$$\begin{bmatrix} X_{11} & X_{12} & \dots & X_{1,1024} \\ X_{21} & X_{22} & \dots & X_{2,1024} \\ X_{31} & X_{32} & \dots & X_{3,1024} \end{bmatrix} \begin{bmatrix} W^k_{11} & W^k_{12} & \dots \\ W^k_{21} & W^k_{22} & \dots \\ \dots & \dots & \\ W^k_{1024,1} & W^k_{1024,2} & \dots \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & \dots & K_{1n} \\ K_{21} & K_{22} & \dots & K_{2n} \\ K_{31} & K_{32} & \dots & K_{3n} \end{bmatrix}$$

Now each row of Q represents a token's need, each row of K represents a token's offer.

For example first row of Q represents the requirement of a second token, first row of K represents what the first token provides. Now what? Now we should compare the two to see if they match! That's the dot product of two vectors, the final number you get measures how different the two vectors are! Using matrix representation, QK^T is doing that. Creating a 3x3 matrix, named 'similarity matrix':

$$\begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix}.$$

Learning0to1

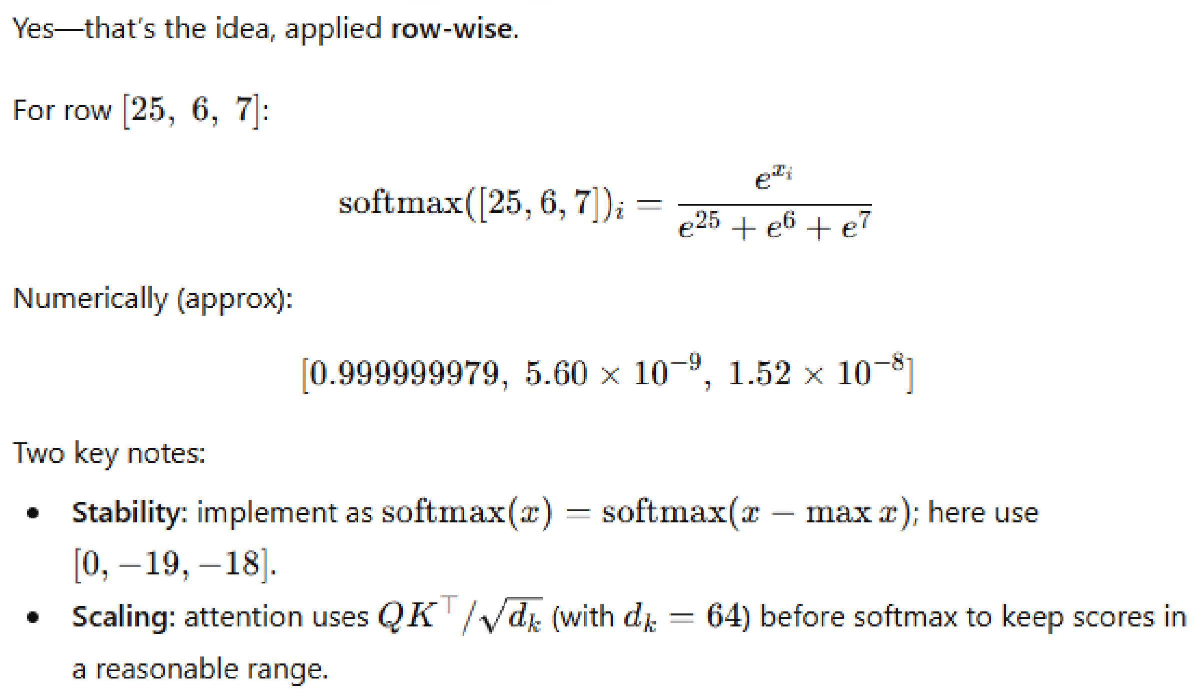
2025-11-13

So two W matrices representing the ‘process’ to ‘analyze’ the token’s meaning in order to figure out their request Q and offer K. **GPT keeps pushing the idea of ‘subspace projection’ to me, which I found weird and incompatible with the overall process here. I mean, once you view everything in terms of geometry ie, a word as ‘a point’, how would you explain ‘adding one point to another’? And what does projection into a subspace even mean? You have so many different W matrices: W_K, W_Q, W_V each one projecting to a subspace, but belonging to a same space? Because to me, a projection of the original space to a subspace means the subspace still belong to that original space, just the dimension got shrunk. But I think K, Q, V have different ‘meanings’ now, so they might have the same dimensions, but they are not the same ‘things’, hence not in the same space! In short, I am not fan of geometrical ideas here.

Once Q and K are figured out, dot product is used to check ~~if they match~~ how good a match is, the end result of the dot product is a measure of the similarity.

But notice that the similarity measure, the dot product has not only cosine angle there, but also the product of two vector’s length! Hence some kind of normalization needed to transform those numbers to between 0 and 1, to become some numbers suitable as weights. For example, one similarity matrix could look like this:

$\begin{bmatrix} 25 & 6 & 7 \\ 5 & 23 & 15 \\ 89 & 30 & 150 \end{bmatrix}$. I think because we want the weights per row wise, hence each row should add up to 1. A simplest way I can think of is just, divide each element by the sum of the row! The softmax seems to be using exponential function to scale each element first, hence the first element of 25 would become $\frac{e^{25}}{e^{25}+e^{6}+e^{7}}$? When I ask GPT it gave me the following answer:



Yes—that’s the idea, applied **row-wise**.

For row $[25,\ 6,\ 7]$:

$$\mathrm{softmax}([25,6,7])_i = \frac{e^{x_i}}{e^{25}+e^{6}+e^{7}}$$

Numerically (approx):

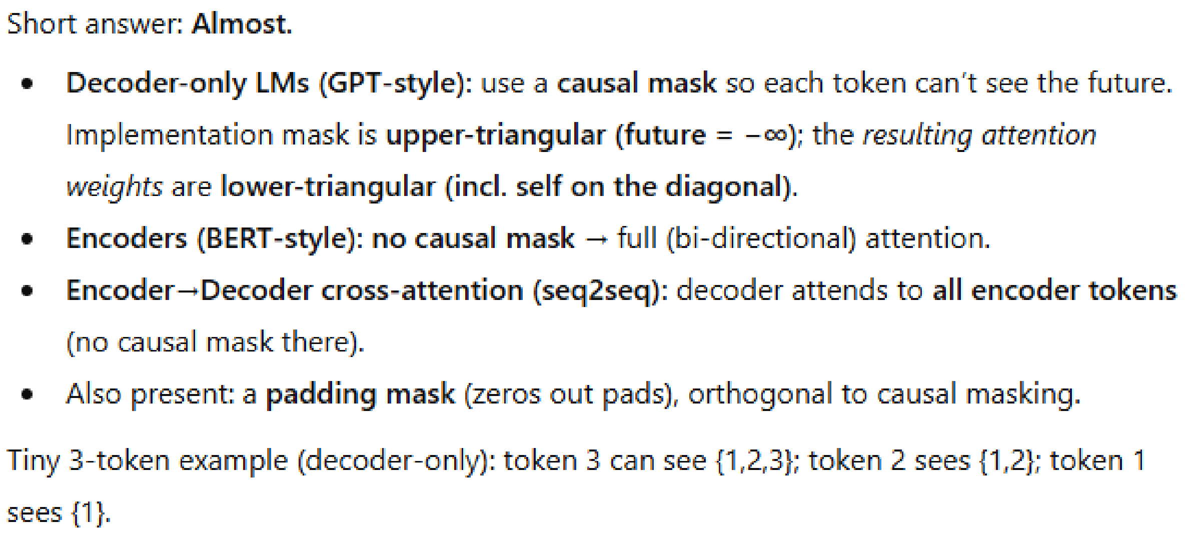
$$[0.999999979,\ 5.60\times10^{-9},\ 1.52\times10^{-8}]$$

Two key notes:

- **Stability:** implement as $\mathrm{softmax}(x) = \mathrm{softmax}(x - \max x)$; here use $[0, -19, -18]$.
- **Scaling:** attention uses $QK^{\top}/\sqrt{d_k}$ (with $d_k = 64$) before softmax to keep scores in a reasonable range.

So the question is, why not straight division over the sum? Why going through exponentiation? This exponentiation tend to create non-linear effects, so that relatively small number becomes extremely small numbers, for example, 6 and 7 become near to zero! Also what does it mean by the stability part, by offsetting each number by subtracting the max? Is there a theory or, purely heuristic? These questions we can park and deal with later, I suspect they are more heuristic.

Then of course, the attention matrix needs to be upper triangle, how to do that? I suspect we can simply discard the upper diagonal terms? Clarifying with GPT, more chaos show up:

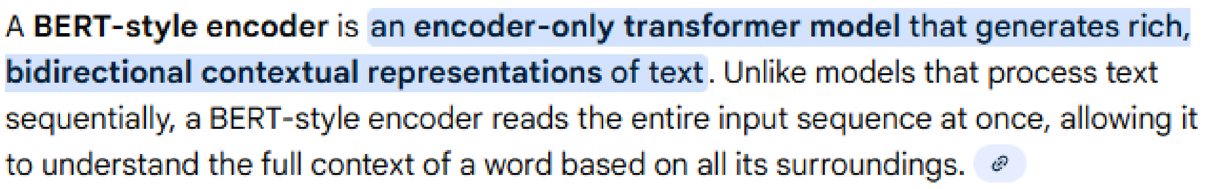


Short answer: **Almost.**

- **Decoder-only LMs (GPT-style):** use a **causal mask** so each token can't see the future. Implementation mask is **upper-triangular (future = −∞)**; the *resulting attention weights* are **lower-triangular (incl. self on the diagonal).**
- **Encoders (BERT-style): no causal mask** → full (bi-directional) attention.
- **Encoder→Decoder cross-attention (seq2seq):** decoder attends to **all encoder tokens** (no causal mask there).
- Also present: a **padding mask** (zeros out pads), orthogonal to causal masking.

Tiny 3-token example (decoder-only): token 3 can see {1,2,3}; token 2 sees {1,2}; token 1 sees {1}.

Decoder and Encoder? Should I ask so to create more chaos? Maybe later. Simple google search shows this:



A **BERT-style encoder** is **an encoder-only transformer model** that generates rich, **bidirectional contextual representations** of text. Unlike models that process text sequentially, a BERT-style encoder reads the entire input sequence at once, allowing it to understand the full context of a word based on all its surroundings.

Interesting. I would think human reading is BERT-style! Because for complex sentences, I do tend to read back and forth!!!

Ok, so now we have the attention matrix, lower diagonal, because GPT is in one direction: so we would have something like this

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.2 & 0.8 & 0 \\ 0.02 & 0.78 & 0.2 \end{bmatrix}.$$

Now these weights can be used to combine words' meanings together to create 'contextualmeaning'. Now what? I would imagine just do the weighted average!

$$w_{11} * t_1$$

$$w_{21} * t_1 + w_{22} * t_2$$

$$w_{31} * t_1 + w_{32} * t_2 + w_{33} * t_3$$

Yes, multiply weights on words (vector of meanings), then boom! But it's not! These weights are not applied to meanings (tokens) directly, but instead, on Value, V, which is again produced by a linear process: $XW_V = V$, V is of the same dimension as Key, and Query, so $n_{token} \times n_p$, 3x64 in this specific case. Attention matrix is going to be applied to this "Value" matrix: AV, A is 3x3, V is 3x64, so we end up with 3x64. But I am wondering why: why the attention (weights) are not applied to the original words directly, but instead on some processed meanings? GPT says: V (What to take): the payload to mix in once a match is found (e.g. verb's tense/number embedding to inform the subject???). I am not exactly sure what it means, but sounds like, the meaning of word represent something, like tense and number, but then what? If it is tense so what, if it is number so what? Hence this V seems to be that, reflecting the action later. So embedding represents the 'being', V represents the action implied. We should dig into more details later. But I am good for now.

So a brief recap, in the 'figuring out contextual meaning' step:

1) Analyze token's need by W_Q: $XW_Q = Q$
2) Analyze tokens' offering by W_K: $XW_K = K$
3) Check how good one token matches another using dot product: $S = QK^T$, similarity matrix S of 3×3, if we there are 3 tokens 5x5 if 5 tokens, nxn if n tokens. Why square? Because it marks the pairwise importance!
4) Then normalize those similarity measurement into weights, using softmax (fancy way to say exponentiation?) So we end up with attention matrix A in 3x3.
5) Apply attention matrix to the Value matrix, AV =Y.

Learning0to1

2025-11-16

A thought just occurred regarding representing words as vectors.

- When should we use vectors to represent 'things'?
- What 'things' should be represented by vectors?
- If not using vectors, what else can we represent things?
- What else can we say about 'using vectors' to represent things?

The starting point is perhaps think about the alternative format to represent things. Then another question emerges: what are we representing? I think we are representing the 'states' of things. Then the first choice is 'scalar', a number. But then, what about function? Sure it represents the mapping, relationship, but this mapping eventual lead to another thing. Function is like a systematic way of crunching scalar. Scalar-valued function we must say.

We view temperature as a variable, denoted by *T*, but we know very well that each *T* will be a number. We view velocity as a vector, denoted by **v.** but wait, why would temperature be a variable? Why would it vary? Vary with what? See the concept of a 'variable' occur when we connect a few dots. Temperature today, temperature tomorrow, temperature the next day, and the next …., they are totally separate things, even though they are all temperature. But once we connect them, deeming them the same thing, but just happening on a different day, they become a variable, one thing: *T(n)***.** Velocity here, there are separate things, but once we connect them by viewing them the 'same velocity just at different positions', we get a velocity-location mapping: **v(x,y),** meaning at different locations (x,y) we would have different velocities.

So why I mentioned temperature and velocity, because we are dealing not with just one temperature or one velocity, but a group of velocities and temperature. Similarly, we are not dealing with one word, but instead a group of words. But what's the difference here? Well in LLM we are not trying to find an overall pattern describing all these words, but we want to predict the next word. But is that a fair statement? Can we not consider words as a time sequence: w(t), and then if we can find an overall formula describing the word at a time, then we could definitely use it to predict the next word! But why we couldn't? Think about stock price prediction model, you can aim to predict the overall formula describing the price at any time t, S(t), or you can adopt the time series model, using a few previous price to predict the next one, very similar to what LLM is doing here.

So from the above rambling, a clearer question emerge: *why we not try to find the overall formula describing the word at a time t w(t), but instead we want to use a few words to predict the next one?*

Also just had a thought regarding using vectors to represent things (words ie.):

1) They are different, that's why they are vectors
2) They are made up of the same components

- That's why math operations still possible, but you need to make sure you are adding same components!