Transformers

Saturday, 12 April 2025 2:29 PM

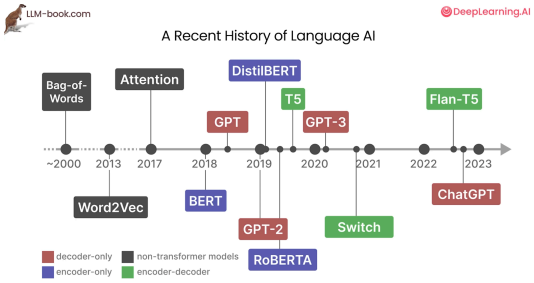
Transformer is a scalable deep learning model that takes seq-to-seq as input and throws seq-to-seq output.
Transformers uses self-attention layer and can be scalable
It has encoder and decoder based Architecture

Main Impact points to consider -

1. Highly scalable and Advance
2. Democratize AI for public use
3. Multimodal capability
4. Acceleration of Gen AI
5. Unification Of Deep Learning

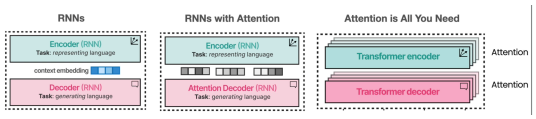
Story Behind Transformer -

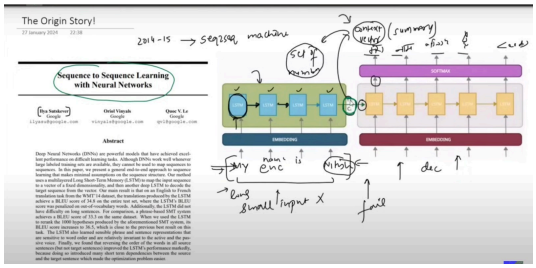
Over the time basic methods such as BOW, Word2Vec are the starting point of Transformer architecture.



1. **BOW - Just considers the count of words and no contextual meaning of words**
2. **Word2Vec - Can create embedding using neural networks. It generates the static embedding regardless of the context.**

History of Improvements to finally arrive at Transformer -

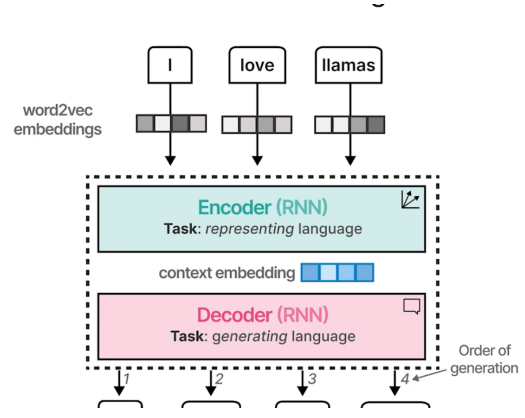




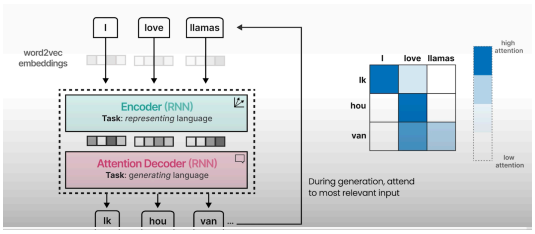
1. **In 2014, a sequence to sequence learning came into existence and that was based on Encoder and decoder technique.**

Both Encoder and decoder were built from LSTM Sequential structures. It is mostly used for Language translation. If you pass a sentence, each word will be process through a LSTM model in encoder and at last it will send a Context vector (summarization of sentence, vector) to decoder. Decoder will process this context vector word by word and then finally translates it to output.

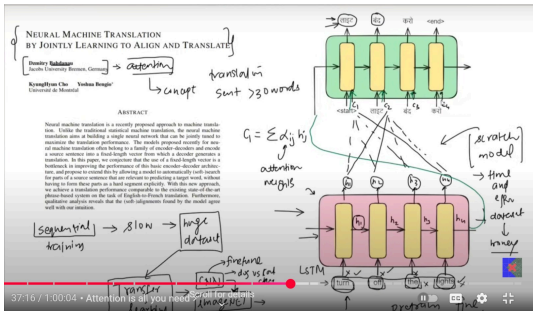
Problem : **Problem behind this paper was, it meant for smaller data and was very slow because it was using seq to seq LSTM. Therefore, training was very slow. Also, if you keep a very lengthy text of let's say 30 words, context vector will not capture context of words properly and hence translation quality started degrading.**



2. **RNN With Attention - Attention is a mechanism to focus of a word relevant to other words. Hence the consecutive words get more importance rather than far words.**
Using Attention entire sequence of embedding can be used by decoder.

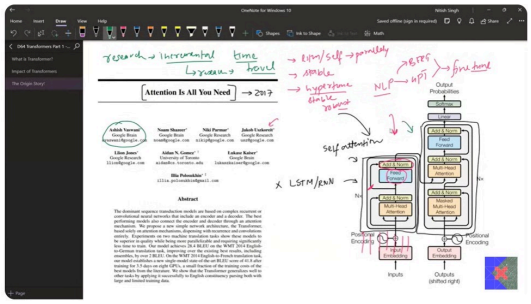


2. 2nd paper is very similar to earlier paper and have same concept of encoder & decoder. Both encoder & decoder were built of LSTM same as last.
Major change was, instead of sending complete context vector at once, in this it started to use the concept of Attention. Here weighted context vector came into picture, which solved the problem of long text translation. However, it still works in sequential architecture & hence training was very slow, therefore it can't be used for scale operations



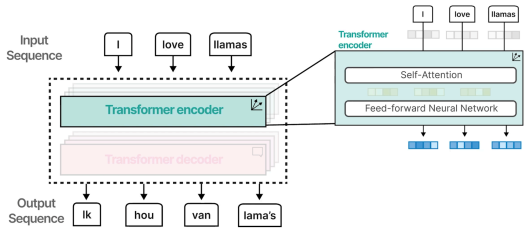
3. In the 3rd paper, Transformer concept released. It also contains encoder & decoder but it does not have LSTM inside it. It uses the concept of self-attention. It is scalable, fast and can use the transfer learning.

Transformer has changed the paradigm of encoder-decoder mechanism. It solves many problem in NLP and large language field. It is the core behind GPT, Bert models.

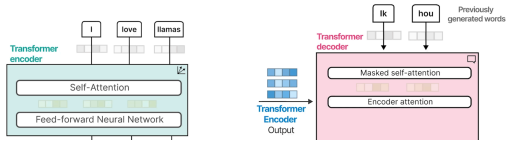


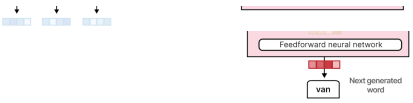
Encoder -

It consist of a self attention layer that generates the contextual embedding for sequence of words and pass to feed Forword neural network.

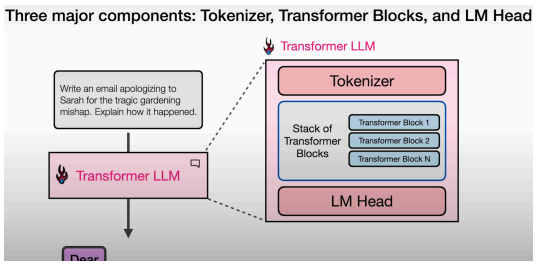


Decoder -
It contains a masked self-attention layer to generate context embedding and pass to encoder attention to use encoder embedding and pass to feed Forword neural network.



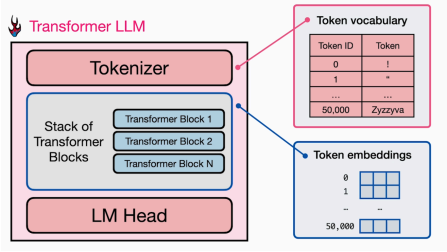


Components of Transformers -

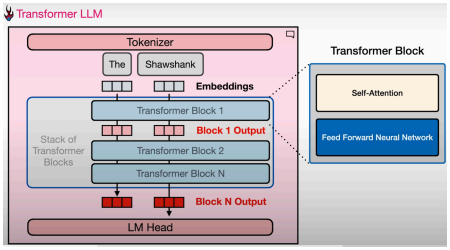


1. **Tokenizer** - Tokenize the textual data into word, sentence or documents, Tokenizer is essentially holds the vocabulary of tokens.

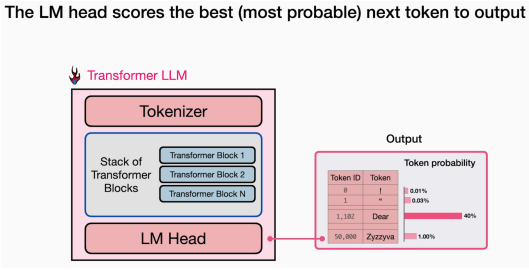
2. **Stack of Transformer blocks** - Stack of Transformer converts these token into contextual embeddings.



Each Transformer blocks have 2 things present -

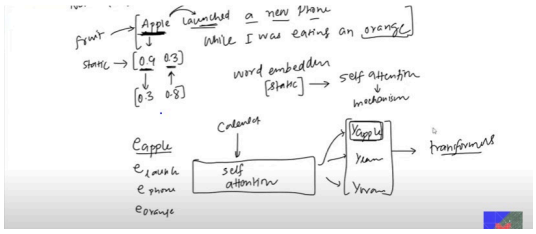


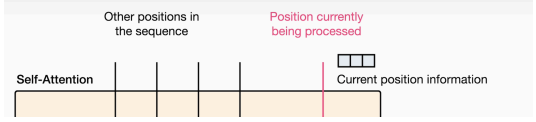
3. **Language model**

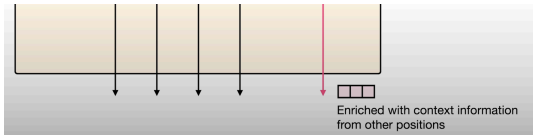


What is self-Attention ?

Self-attention is a mechanism in which it generates the smart contextual embedding.
It takes static embedding as an input and generate smart contextual embeddings that is used in the Transformers







It basically takes the previous token also in consideration while generating the contextual embedding.

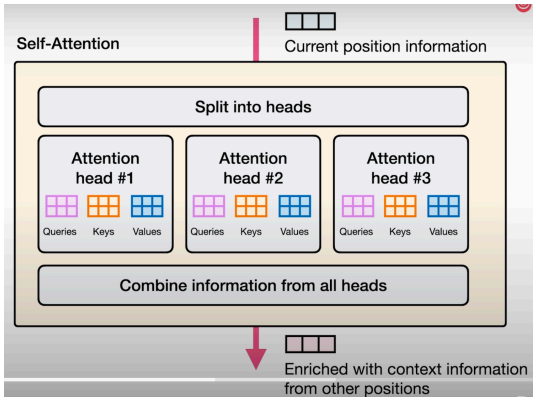
Self Attention is formed on 2 mathematical components -

1. **Relevancy score**
2. **Combination of Information**

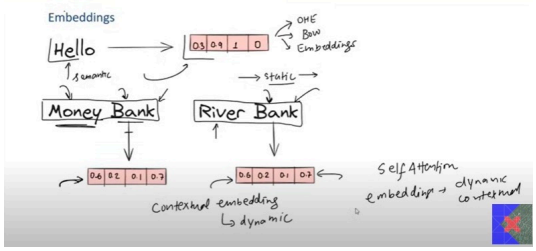
Relevance score is calculated by each of the Attention head where Query vector multiplies with the Key vector and each tokens generates some weights in overall context of that sentence.

Self-Attention have lots of attention heads which runs in parallel.

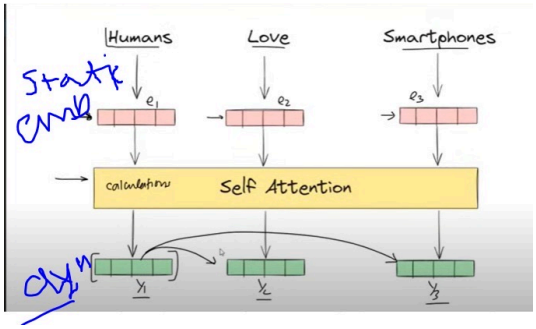
Combination - combining all the information from all the attention heads.



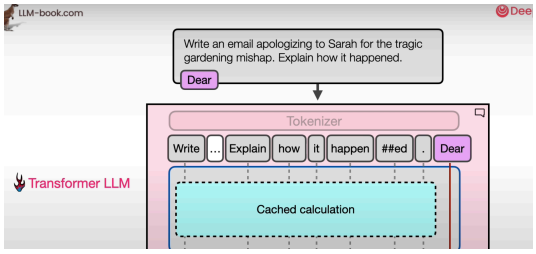
Contextual embeddings are embeddings that changes with the context and can't be static like general word embedding.

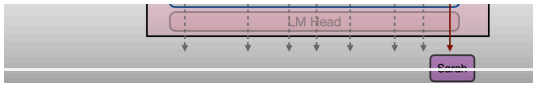


Self-attention is a way to covert static embedding to dynamic contextual embeddings.

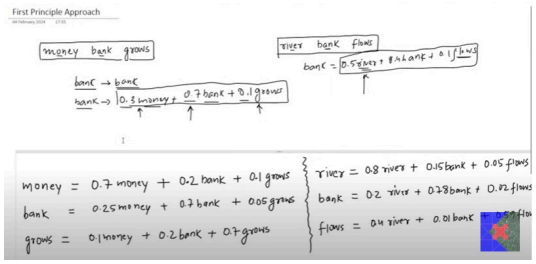


KV (Key Value) Cashing - In the older methods when generated next word (token) passed as an input all calculation has be to redone which essentially takes longer time to compute but here caching of older calculation possible in matrix format and hence long context can be catered.

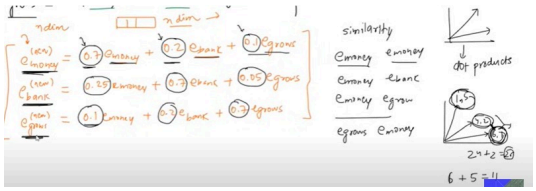




Understand with Example :



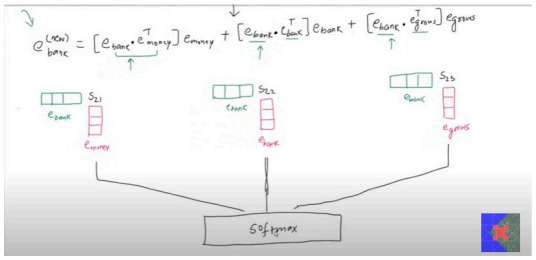
In Statement 1 and Statement2 **Bank** word represents different meaning, therefore if we use the same static embedding it will have a mislead information.
Now, let's assume if we can use each word not as an individual word but with the context of other word as shown in the image. Bank context changes with statement



Words essentially represented by embeddings in computer.
So we are taking the weights of each word embedding to generate the new contextual word embedding.

What does this weights signify and what happens when we multiply these weights to embedding ?

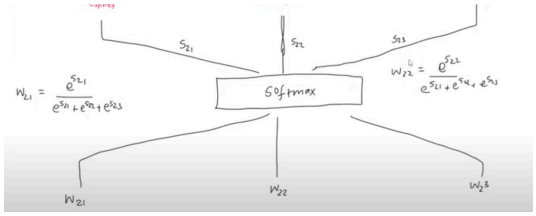
These weights shows the similarity, for example embeddings of money * 0.7 * embeddings of money or embedding of money * 0.2 * embedding of bank, represents a dot product which represents similarity



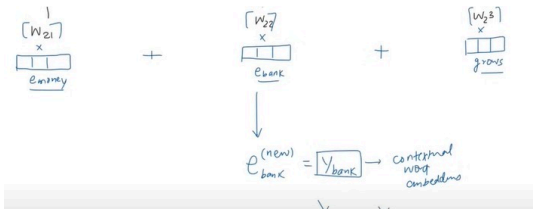
Above Images, shows how new embedding of bank generates after multiplying with other word embedding in same sentence.

When these 2 embedding multiplies using dot product it generates a number let's call it S21 (statement 2 first word) and following S22, S23.
S21, S22, S23 can be any number but we understand these weights must add to 1 therefore we need to normalize it so that we can say bank word has 20% influence from money, 70% from bank and remaining 10% from grows.

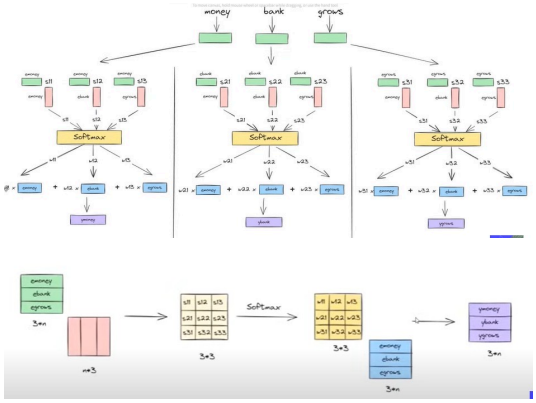
To Normalize we are putting it inside SoftMax function.



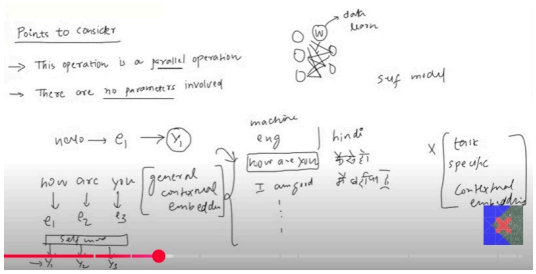
So now once we got the got, multiply with each words embedding as represented in the equation to generate the final contextual embedding.



Similarly we create for Money, grows etc.



These operation can be performed in parallel with the help of algebra, as shown in the image.



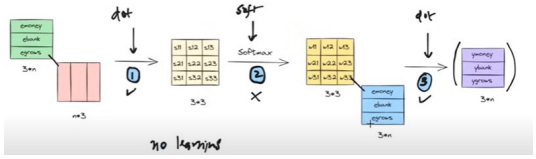
Here is 1 catch, these contextual embeddings are very general and depends on that sentence. It don't care about other data in that whole job. Also, it does not have any learning function which can be utilized anywhere.

Take an example :
Piece of Cake - if you convert in Hindi it shows it's a small portion of cake because that's how we generated our generalized contextual embedding

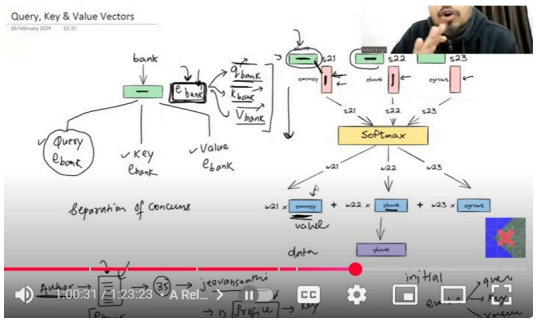
However, in our whole docs piece of cake can be act as very easy task

So we need task specific contextual Embedding

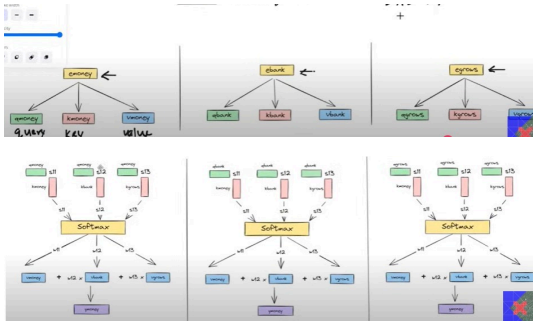
That means we also need to learn while doing the contextual translation by including weights and bias



Now question comes where should we introduce weights & bias to make parameter learnable ?
It is possible only in step 1 & 3

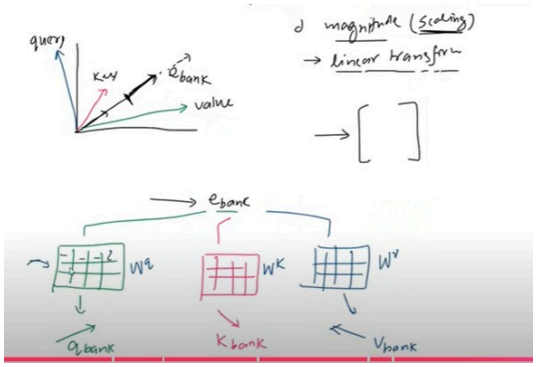


In whole process, a word embedding interacts 2 times, at first it multiplies with itself and at last post weight generation it again multiplies with its original value.
It can be separated for 3 different role shown in image

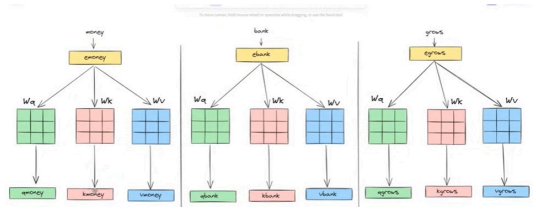


The above images changes with these separated embedding, rather than a single same embedding in all 3 places.

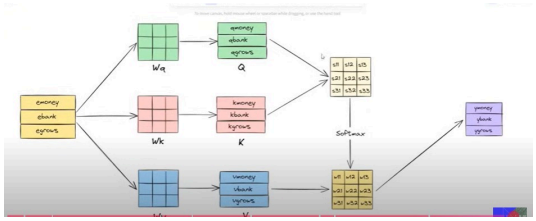
Now how to separate a embedding into 3 new embedding ?



We can only get new vectors from original vectors through Matrix multiplication, so we starts initially with random weight matrix and thereafter, through back proportion, we minimize the loss and optimize the weights

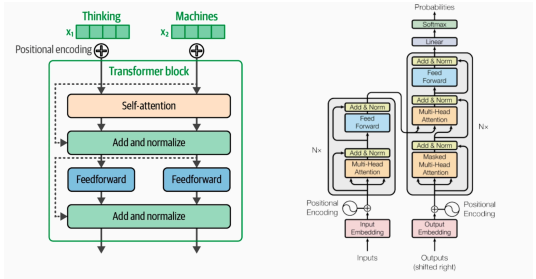


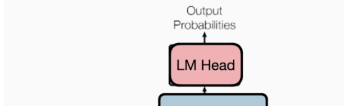
This whole piece can be parallelize and summary view of same in the below image.

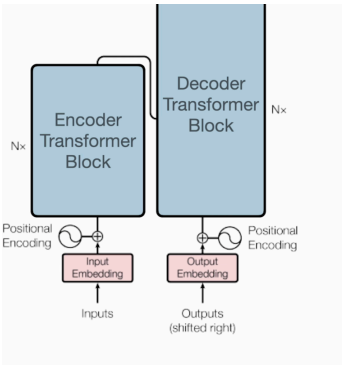


Optional -

Transformer Architecture -







Recent Improvements - Used Grouped Query and Key, shared across multiple attention heads rather than each attention heads contains separate key, query and value embedding.

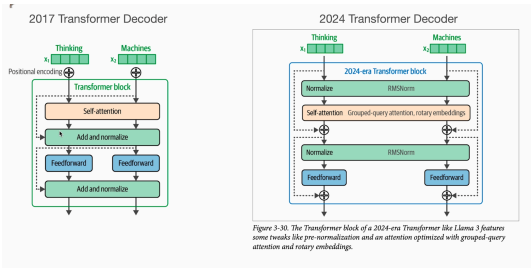
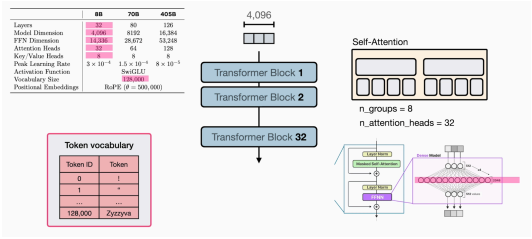
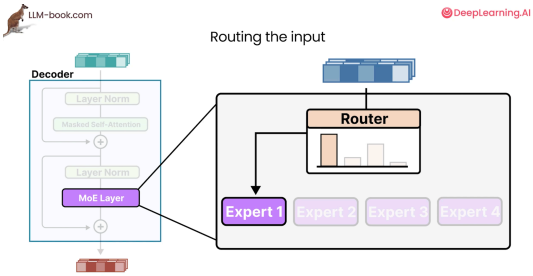


Figure 3-30. The Transformer block of a 2024-era Transformer like Llama 3 features some tweaks like pre-normalization and an attention optimized with grouped-query attention and rotary embeddings.

How to interpret -



Mixture of experts-



Source : https://jalammar.github.io/illustrated-transformer/
Source : https://arxiv.org/pdf/1706.03762
Source: https://www.deeplearning.ai/short-courses/how-transformer-llms-work/?utm_campaign=handsonllm-launch&utm_medium=partner