If we have our data in csv, we can use some code by hugging face, that converts data to hugging face dataset format

STATIC - EMBEDDING/ENCODING:

One Hot Encoding

Freq/Count based:

BOW

TF/IDF

N-GRAM

DP Learning based./Prediction based Embedding:

WORD2VEC(Based on set of features it produce weights to each token. Which holds semantic meaning.

Which gives average weight based on all the sentence)

This is static vector for each Token/word. and produces vector based on Overall conext of the whole data

Contextual/Dynamic vector was no there in this embedding. So Vector going to be same throught the process

CBOW

SKIP GRAM

Issue of WORD2VEC:

But this holds semantic meaning but holds static info. If Bank as river bank or Financial bank both will have same vector

To solve this issue Dynamic embedding, self attention/Transformer arrived. Embedding should change based on sentence for same word

Dynamic/CONTEXTUAL EMBEDDING:- Advnavced EMBEDDING:

Self attention inside Transformer

Which creates embedding Dynamically

MODELS:

ANN

CNN

RNN

Gradient vanishing issue for longer sentence and not much symantic meaning for longer sentence

LSTM

Memory retained but More complex than RNN and slowr than RNN ,

I/P Data need to be passed sequentially or serialy one after the other in next timestamp. so current timestamp had dependency on prev timestamp

it has to do time stamp pased sequetial operation, so it was slow and complex

GRU

ENCODER-DECODER

With RNN/LSTM inside Encoder-Decoder. It passes each encoder's RNN/LSTM output passed to Decoders's RNN/LSTM in sequesnce pattern(Not send to all RNN/LSTM od decoder)

ENCODER-DECODER - WITH ATTENTION

With RNN/LSTM inside Encoder-Decoder, has NN which passes each encoder's RNN/LSTM O/P to Each decoders input as context text)

Which handles Asynchronous data

Issue:

More Complex- Computational complexity due to lengthy sentence due to huge NN to pass each Encoder's RNN/LSTM output to decoders ALL RNN/LSTM

Here Attention also holds some type of Embedding. In Encoder to Decoder context text passing NN, Last layers weight is going to be embedding for that Word/Token.

Which will be becomes I/P for Decoder

SELF ATTENTION(TRANSFORMER)

Which dont has any RNN/CNN

---------------------------------------------------------

SELF ATTENTION(TRANSFORMER)

It has concept - Self attention. Contains Encoder and Decoder

It wont have RNN,LSTM, GRU which works on sequence on timestamp. Sequential processing

But Transformer has only Neural Network, It wont works on timestamp basis. It does parallel processing.

It identifies Embedding info within its same sentence before passing to Decoder. So its parallel processinng , not the seq timestamp based processing

Transformer has multimodel capabilty, can do image/video/audio related or text related task

both are developed to solve seq to seq problem (like transaltion, Machine transalation, Q&A, Summarization)

**Encoder decoder with attention:**

Here encoder's all RSTM/GRU output/Context vector sends to input of all Decoders input of RSTM/GRU via set of neaural network

This network called as attention layer

To handle Asynchronas data(i/p and o/p different length) Encoder decoder concept came.

Then Encoder decoder with attention came to give more importance to specfic feature(Symantic meaning)

Dynamic Contextual embedding/vector, to avoid static embedding in word2vec and encdoder/dercoder and ED with attentiona

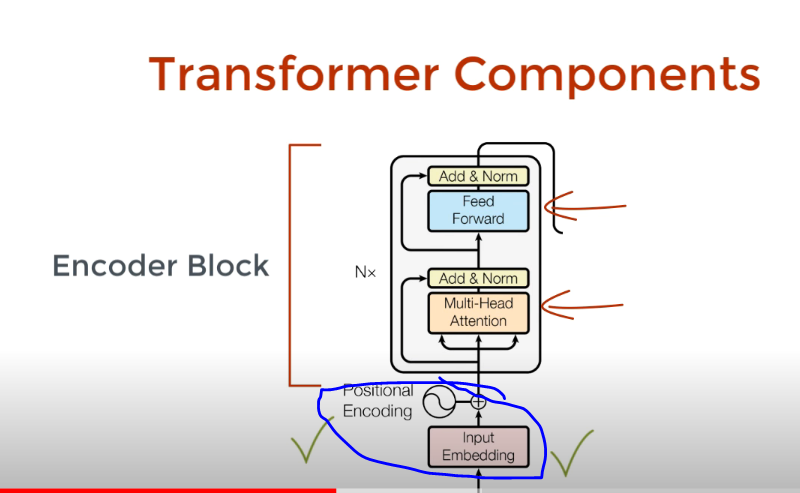
but e/de with attentian had comptation expensive due to long length

Transformer dont need word2vec transformer, its does self attention , that way it converts words to vector.

This type of embediing is called "General Contextual Embedding"

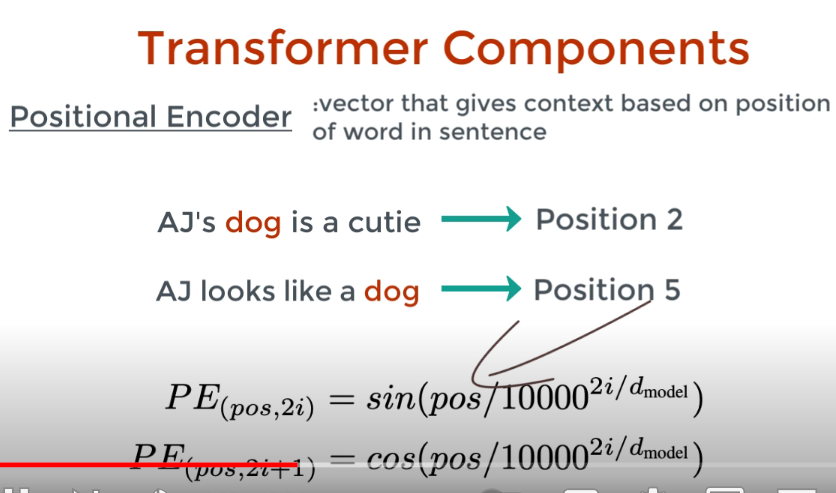
some task we need to have task specific embedding, that time we use parameter.

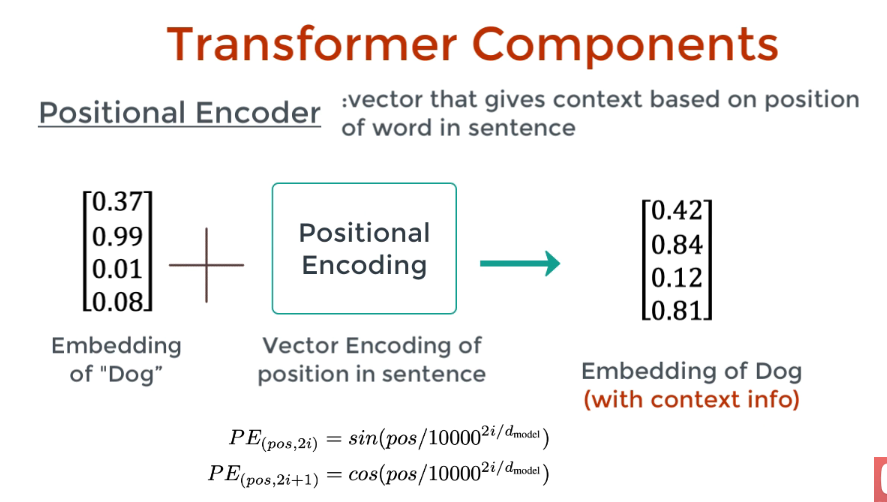
This parameter called learning parameter

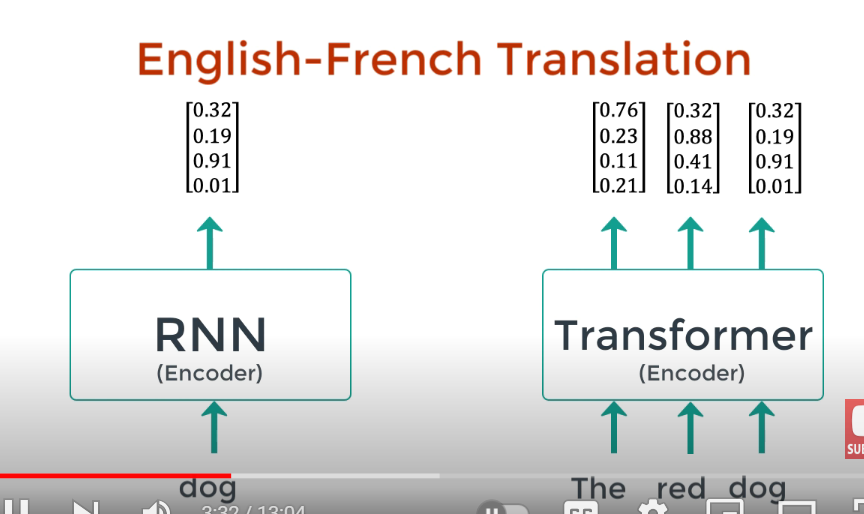


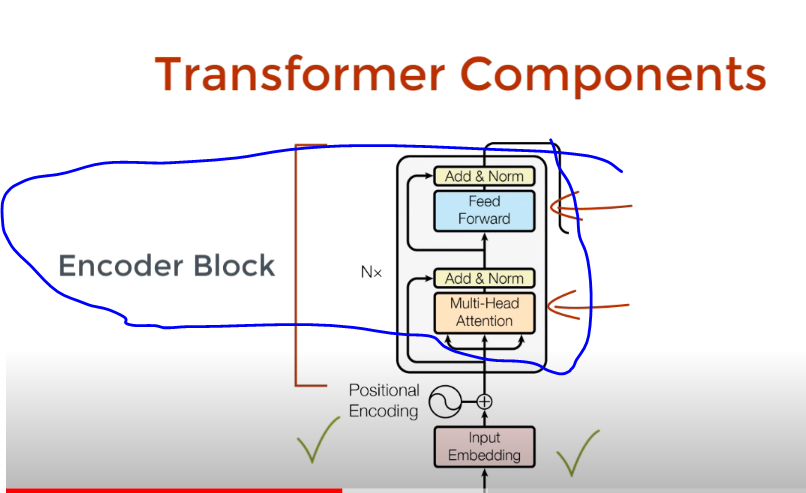
Positional Encoder:

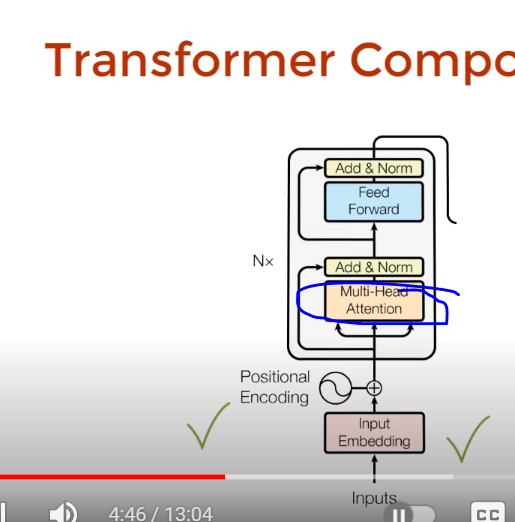
Where parallel processing allowed not time stamp based. Each word will have dependency by other words in that sentence. So meaning of that word depends on the position of that word

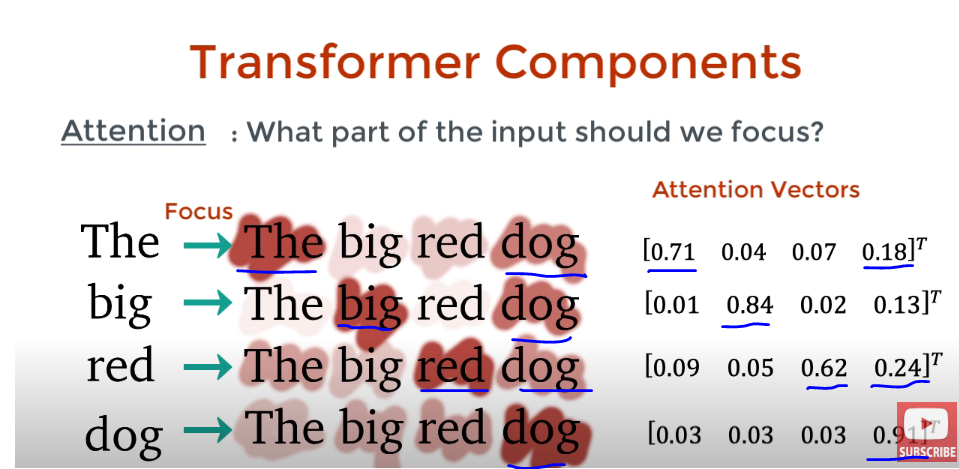


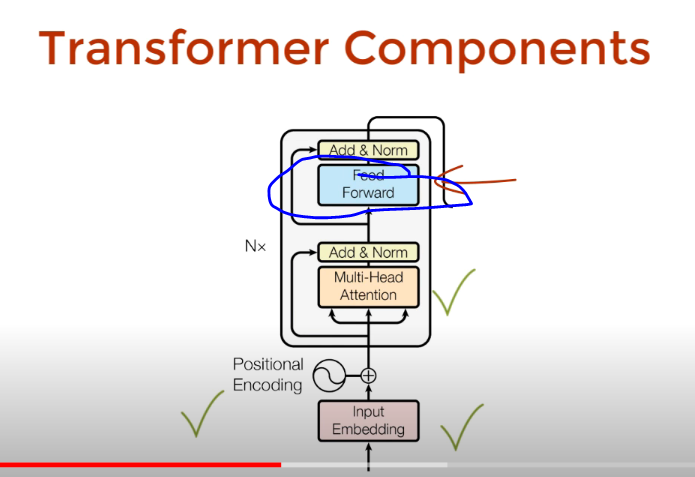


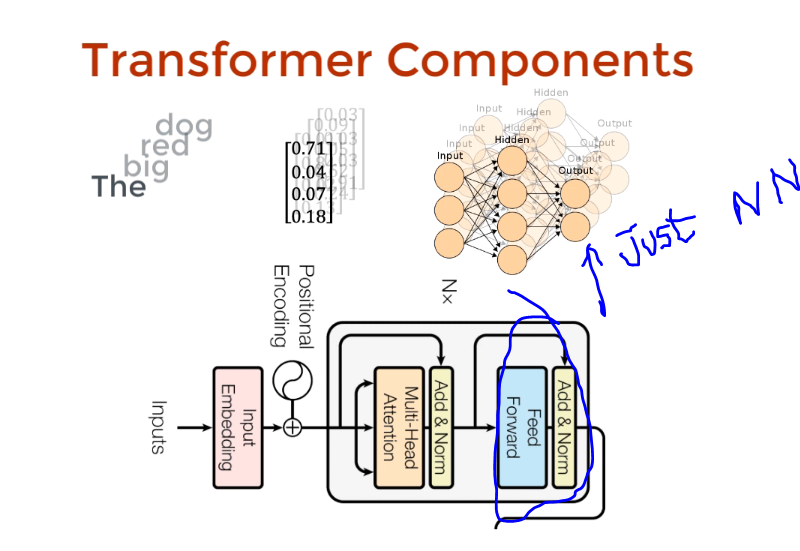


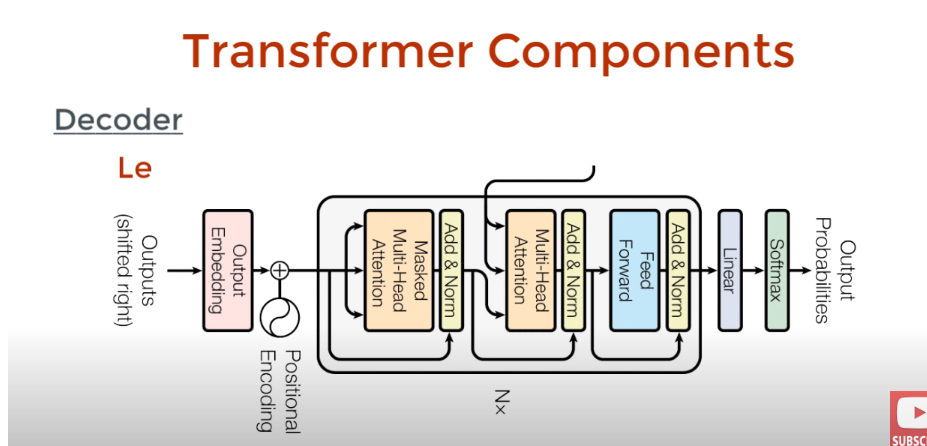


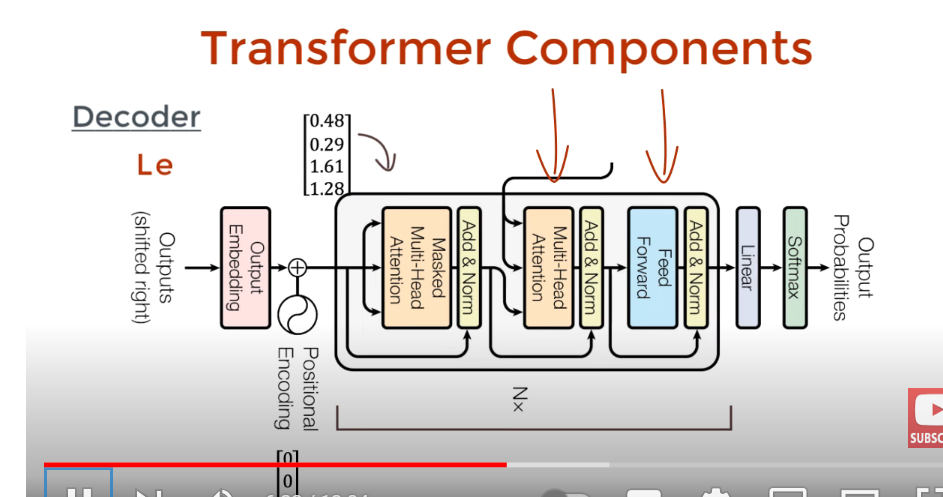


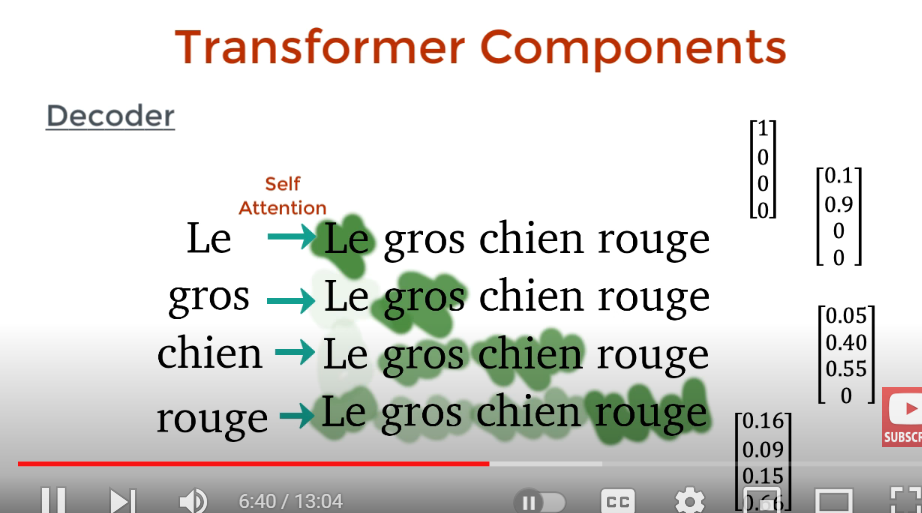




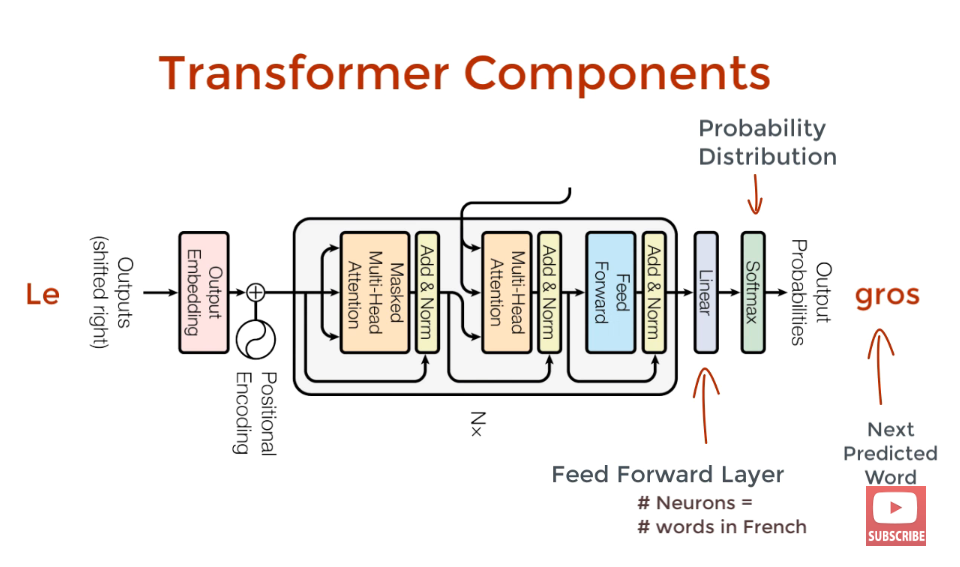


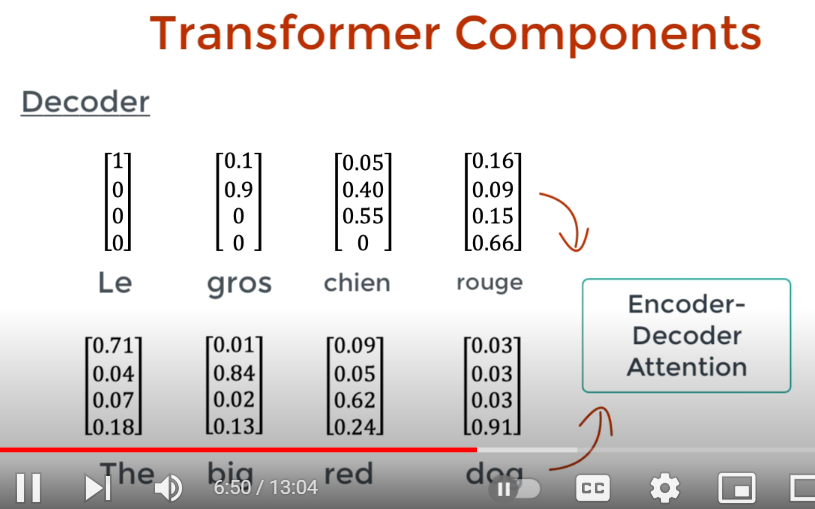




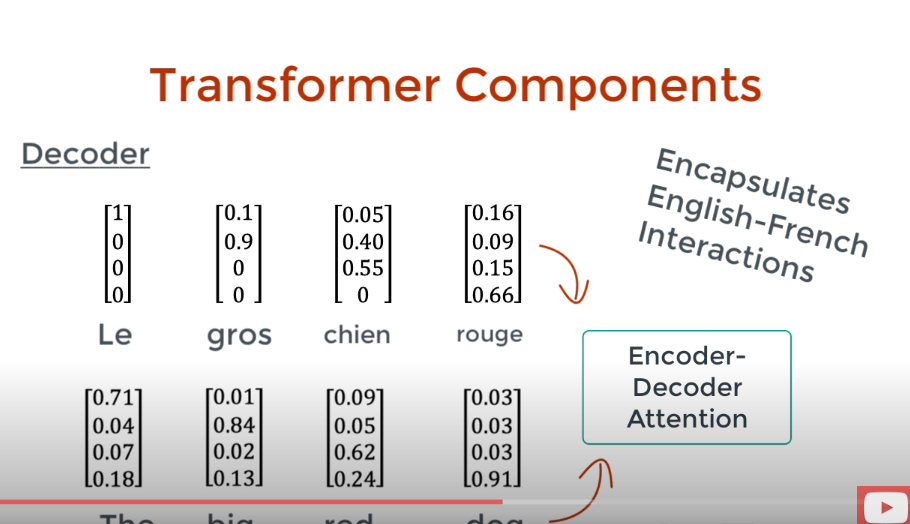


These decoders attention vector and encoders attention vector are passed to next encoder-decoder attention. This finds relationship between english and french words

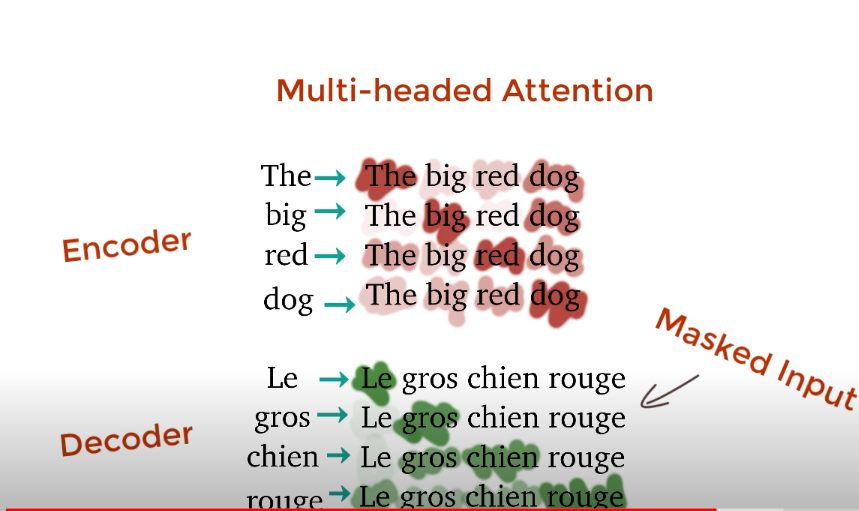


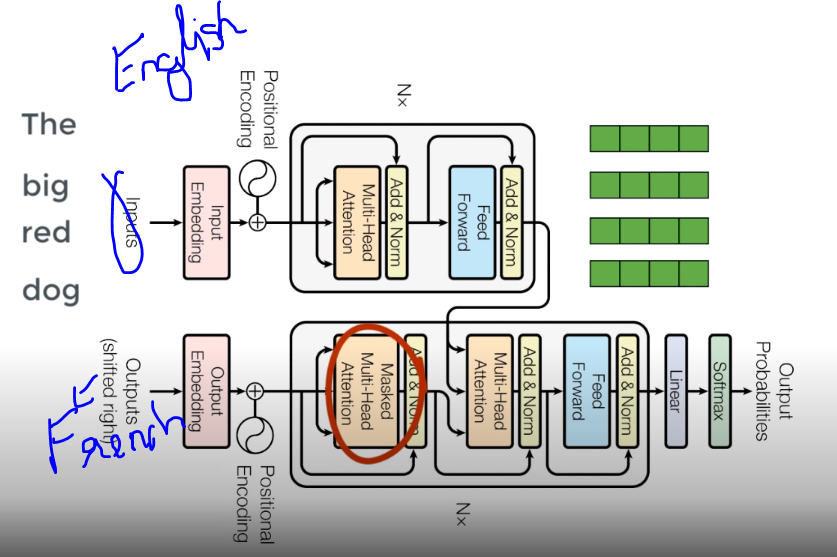


**DECODER:**

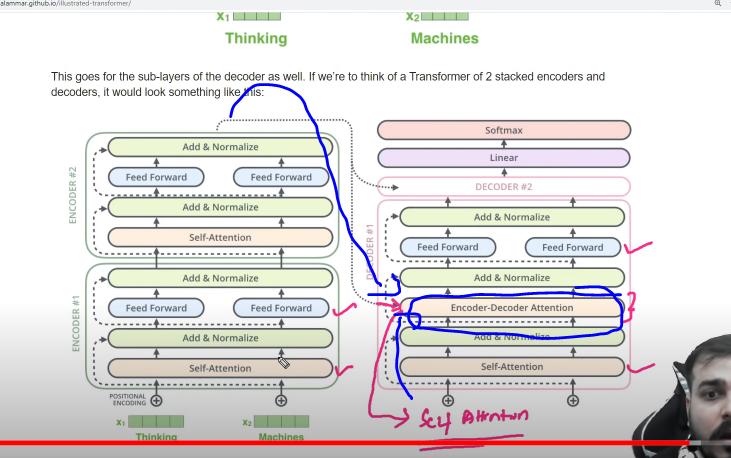


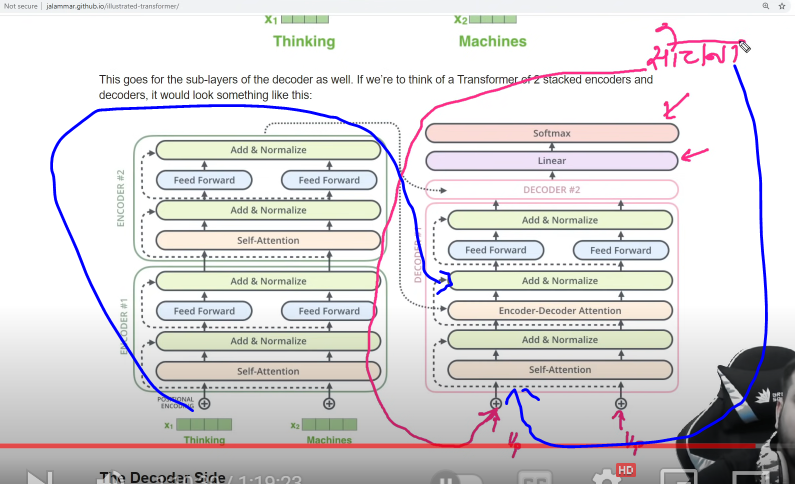
Actually encoder will be fed with English word and decoder will be fed with French word(Like a label in language translation) during training. But decoders french word will be masked to predict next french word based on english word. All the word from english used but only prev word from french word passed in decoder sequentially to predict next french word.





Here only french 1st word **Le** passed as positional encoder vector from decoder I/P along with whole english sentence vector from encder O/P to “**Encoder-Decoder Attention**” inside Decoder, So later it predicts next French word **chein**

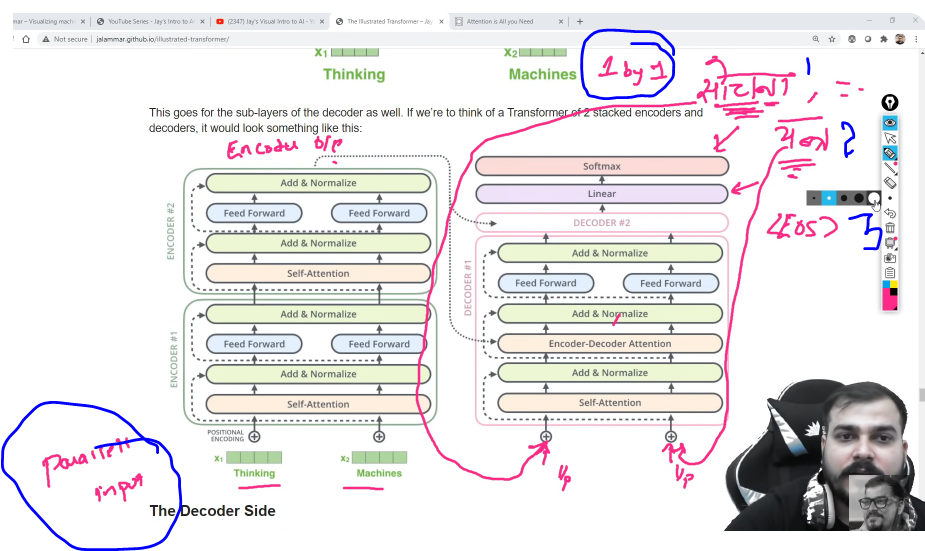


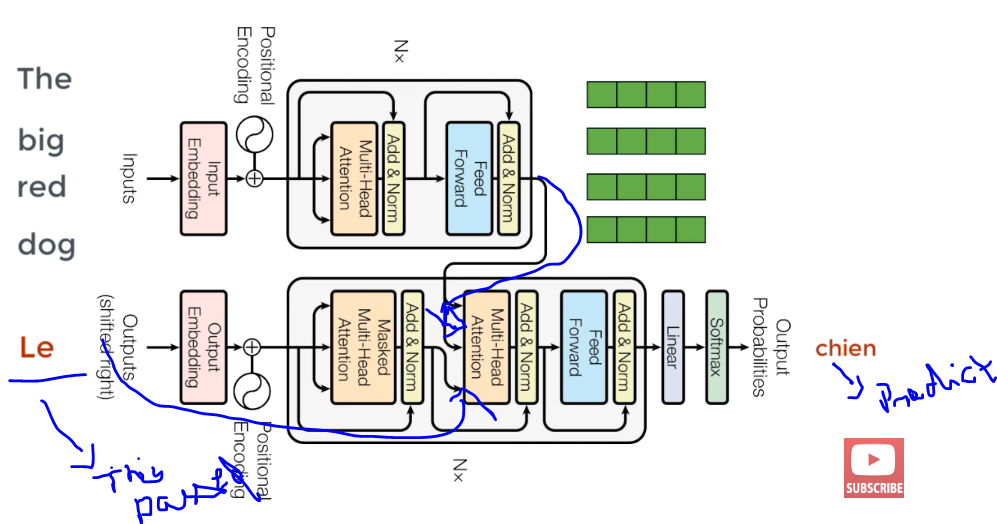


Here in Encoder all the word processed parallel, but in Decoder, each word processed serially/sequentially. Decoder will have 2 Inputs.

1. Encoders all words together as vector
2. Decoders prev time stamp predicted value will be passed as input to Decoder

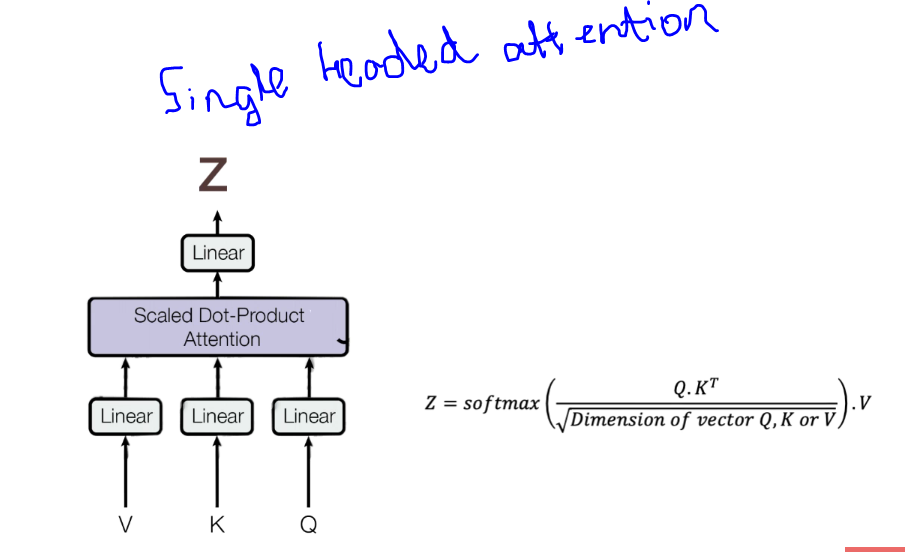
Ex: This 1st predicted Sochana word passed as I/P (2nd input) with encoders Whole input(1st I/p) lines all vector to predict next Hindi word Yntra in hindi . Same way in 3rd iterantion all predicted word will passed as I/P along with Encoders Vector to predict next hindi word. Same way whole process continues till we get EOS





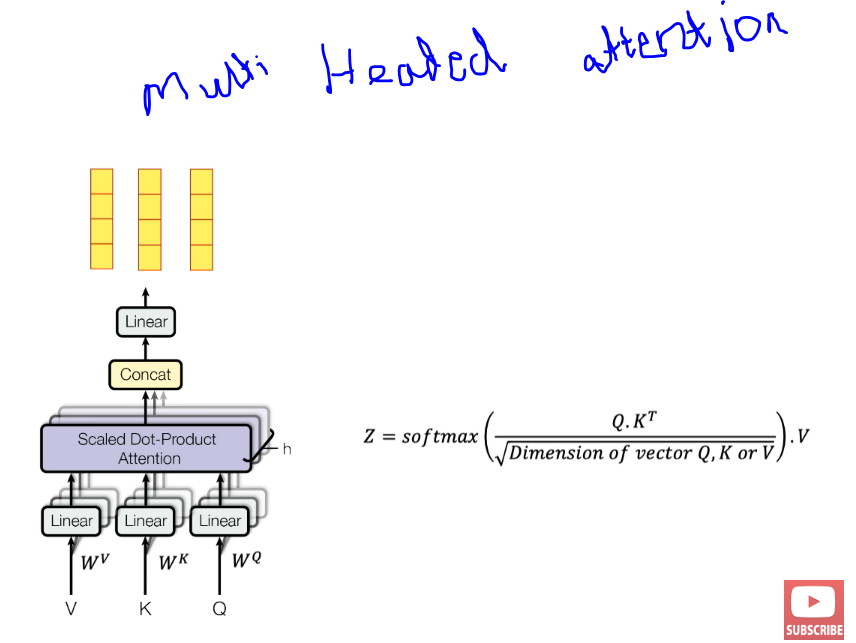
**Single Headed Attention:**

Only single same Weight (Wq, Wk, Wv) are multiplied to all words. So output of encoder z will be 1 vector only. But it wont capture relation of 1 word with other words within that sentence. Only single words relation will be captured

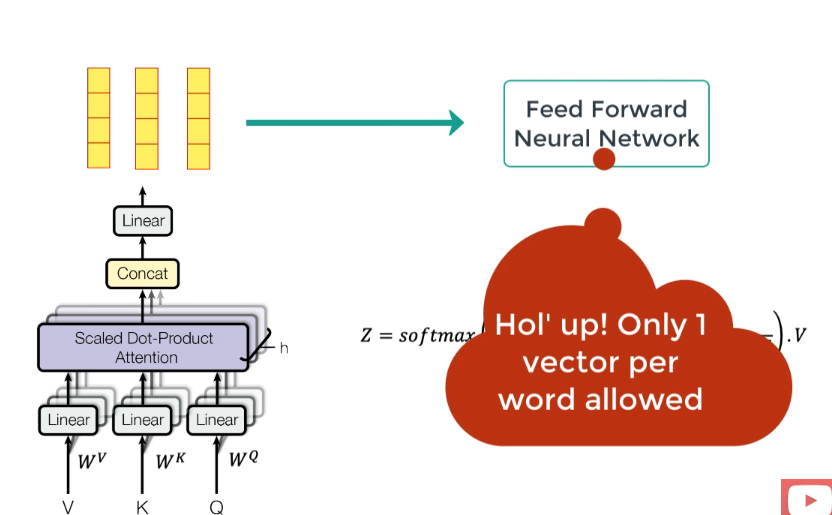


**Multi Headed Attention:**

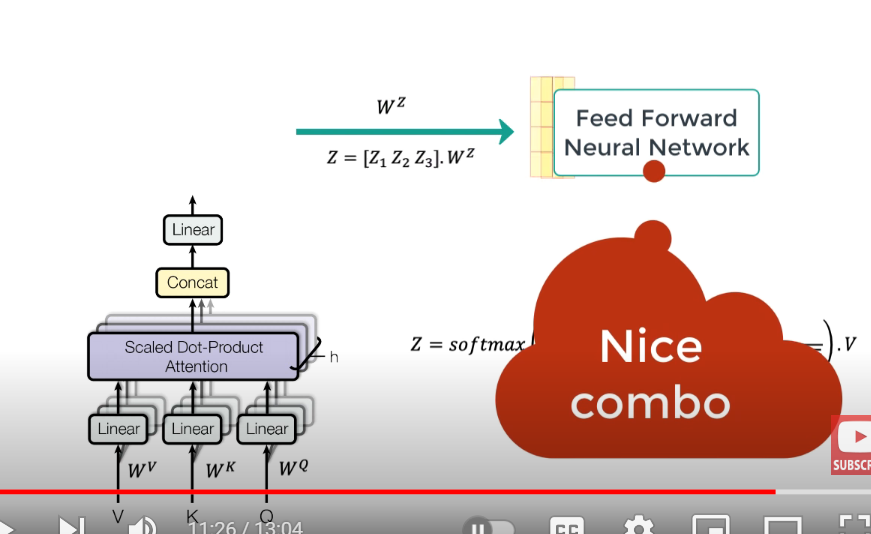
In Transformer architecture researcher used 8 attentions, means 8 different Weight combination (Wq1,Wq2..Wq8, Wk1, Wk2..Wk8, Wv1,Wv2..Wv8) are multiplied to all words. So output of encoder z will be 8 vector . It capture relation of 1 word with multiple words within that sentence due to 8 different attentions/weight



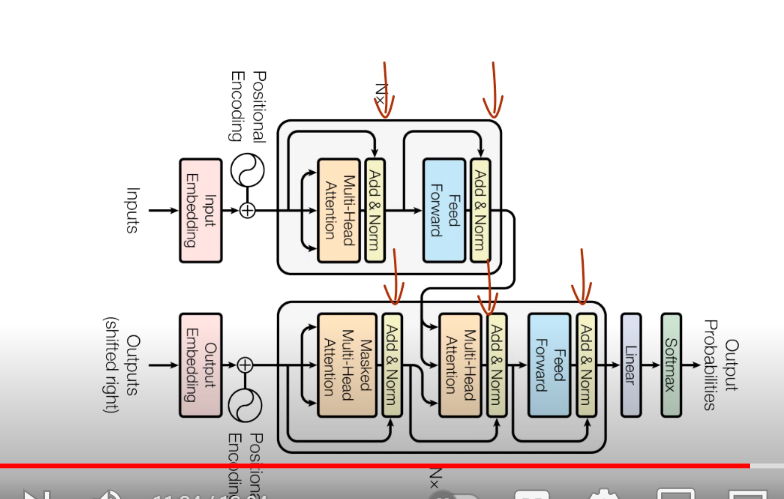
But this multi headed attention creates multiple vector, but NN network except only 1 vectot per word.

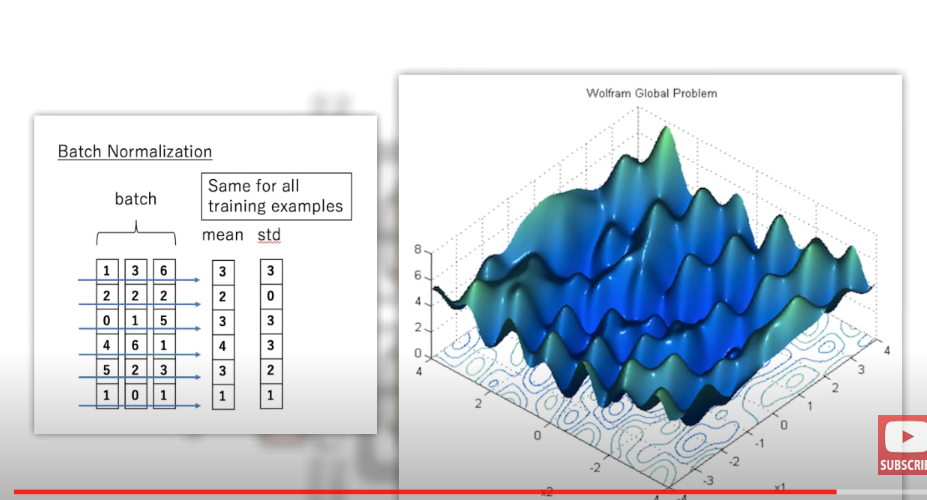


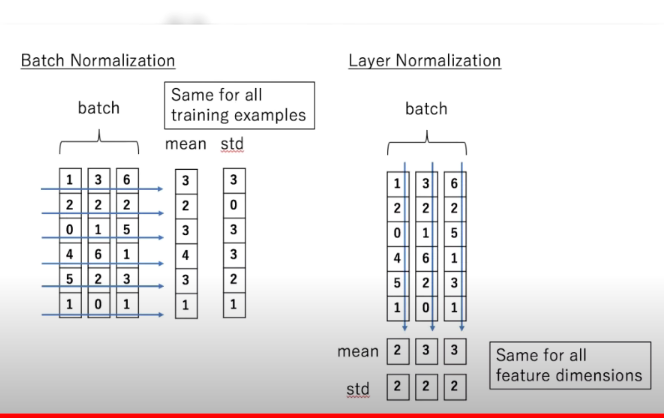
So we used another Weight WZ to make as 1 weighted attention vector per word, which fed as input to feed forward NN



After each layer, we do normalization.







This type

--- Listen 18th feb record from middle-> Transformer

listne 10 th Q anS -->Done

d11th Beginnning and QA -->Done

read some transfrmer video

Download doc/notes

sg=0 will do cbow in costum embedding model

What are RAG?

Here is an example of a simple RAG based Chatbot to query your Private Knowledge Base.  
  
First step is to store the knowledge of your internal documents in a format that is suitable for querying. We do so by embedding it using an embedding model:  
  
𝟭: Split text corpus of the entire knowledge base into chunks - a chunk will represent a single piece of context available to be queried. Data of interest can be from multiple sources, e.g. Documentation in Confluence supplemented by PDF reports.  
𝟮: Use the Embedding Model to transform each of the chunks into a vector embedding.  
𝟯: Store all vector embeddings in a Vector Database.  
𝟰: Save text that represents each of the embeddings separately together with the pointer to the embedding (we will need this later).  
  
Next we can start constructing the answer to a question/query of interest:  
  
𝟱: Embed a question/query you want to ask using the same Embedding Model that was used to embed the knowledge base itself.  
𝟲: Use the resulting Vector Embedding to run a query against the index in the Vector Database. Choose how many vectors you want to retrieve from the Vector Database - it will equal the amount of context you will be retrieving and eventually using for answering the query question.  
𝟳: Vector DB performs an Approximate Nearest Neighbour (ANN) search for the provided vector embedding against the index and returns previously chosen amount of context vectors. The procedure returns vectors that are most similar in a given Embedding/Latent space.  
𝟴: Map the returned Vector Embeddings to the text chunks that represent them.  
𝟵: Pass a question together with the retrieved context text chunks to the LLM via prompt. Instruct the LLM to only use the provided context to answer the given question. This does not mean that no Prompt Engineering will be needed - you will want to ensure that the answers returned by LLM fall into expected boundaries, e.g. if there is no data in the retrieved context that could be used make sure that no made up answer is provided.  
  
To make it a real Chatbot - face the entire application with a Web UI that exposes a text input box to act as a chat interface. After running the provided question through steps 1. to 9. - return and display the generated answer. This is how most of the chatbots that are based on a single or multiple internal knowledge base sources are actually built nowadays.  
  
As described, the system is really just a naive RAG that is usually not fit for production grade applications. You need to understand all of the moving pieces in the system in order to tune them by applying advanced techniques, consequently transforming the Naive RAG to Advanced RAG fit for production. More on this in the upcoming posts, so stay tuned in!

