BERT (Bidirectional Encoder Representations from Transformers)
- (by Google AI Language)

1. Used for NLP tasks: Transfer learning (Fine Tuning)
   (information learned from one dataset can be transferred to other datasets for specific tasks.)

Possible methods for transfer learning:
https://blog.insightdatascience.com/using-transfer-learning-for-nlp-with-small-data-71e10baf99a6
Advantage of BERT: Greater accuracy when performed on smaller data. Whereas getting data is the challenging part today.

2. It's a Bidirectional training of Transformer (**Transformer is forward trained**)
   (for more detail of transformer check: http://jalammar.github.io/illustrated-bert/)
3. It's a popular **attention model for** language modelling.

Attention mechanism: In psychology, attention is the cognitive process of selectively concentrating on one or a few things while ignoring others.

(**Difference between autoencoder and encoder-decoder**:
Auto-encoder is a neural network with a single hidden layer and the number of nodes in the input and output layers. In encoder-decoder, there are two networks namely the encoder which 'encodes' the input data to a latent space (the **latent space** is simply a representation of compressed data in which similar data points are closer together in **space**) and a decoder that 'decodes' it to data point from the distribution of the training data.

Learning happens in both by minimising the loss function using backpropagation. But once an encoder-decoder is trained, the encoder part can be kept aside and we can supply noise sampled from any distribution as a latent vector to the decoder to generate data points. This is not possible in autoencoders. Auto-encoders are used to learn the representation of data in a lower dimension)

Before the technique was based on the encoder-decoder RNNs/LSTMs technique, The encoder LSTM is used to process the entire input sentence and encode it into a context vector. Whereas decoder LSTM or RNN units produce the words in a sentence one after another.
Drawback: The performance of the encoder-decoder network degrades rapidly as the length of the input sentence increases (vanishing/exploding gradient problem)
To overcome the above problem attention mechanism was introduced.
Bidirectional LSTM used here generates a sequence of annotat*ions (h1, h2,….., hTx)* for each input sentence. All the vectors h1,h2.., etc., used in their work are basically the concatenation of forward and backward hidden states in the encoder. All the vectors h1,h2.., etc., used in their work are basically the concatenation of forward and backward hidden states in the encoder. The context vector ci for the output word yi is generated using the weighted sum of the annotations. The weights are computed by a softmax function.

Example:
"I am doing it". Here, the context vector corresponding to it will be:

C=0.2*I"I" + 0.3*I"am" + 0.3*I"doing" + + 0.3*I"it"
[Ix is the hidden state corresponding to the word x]

https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/

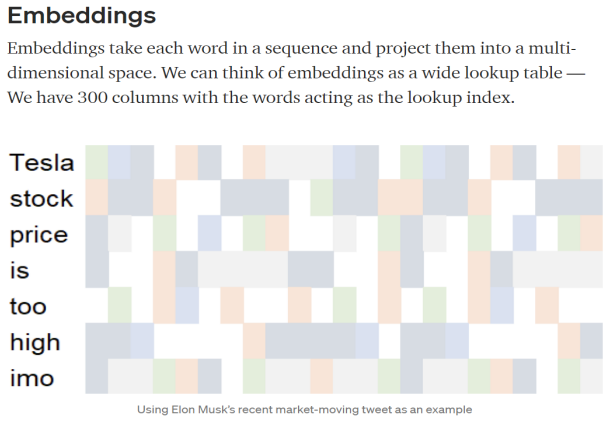
Working of BERT:
BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary.

Benefits of using BERT:
When training language models, there is a challenge of defining a prediction goal. Many models predict the next word in a sequence (e.g. "The child came home from ___"), a directional approach which inherently limits context learning. To overcome this challenge, BERT uses two training strategies:

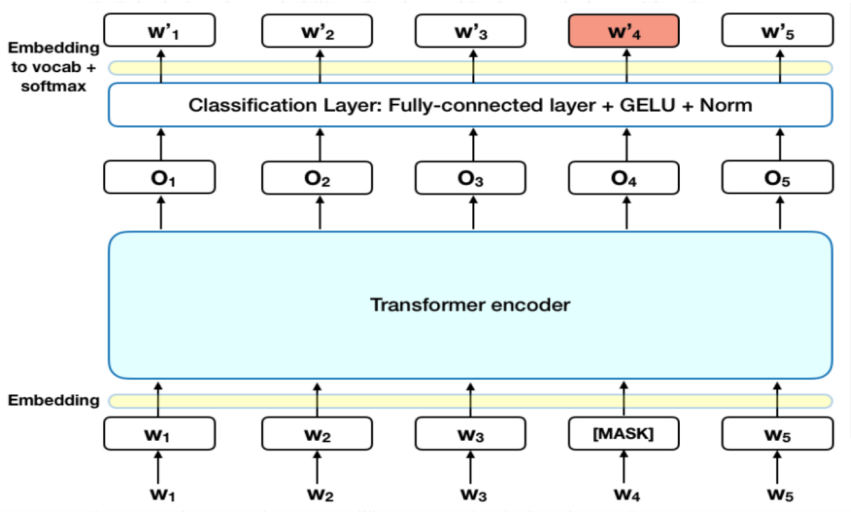
Masked LM (MLM)

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

1. Adding a classification layer on top of the encoder output.
2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculating the probability of each word in the vocabulary with softmax.



The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness

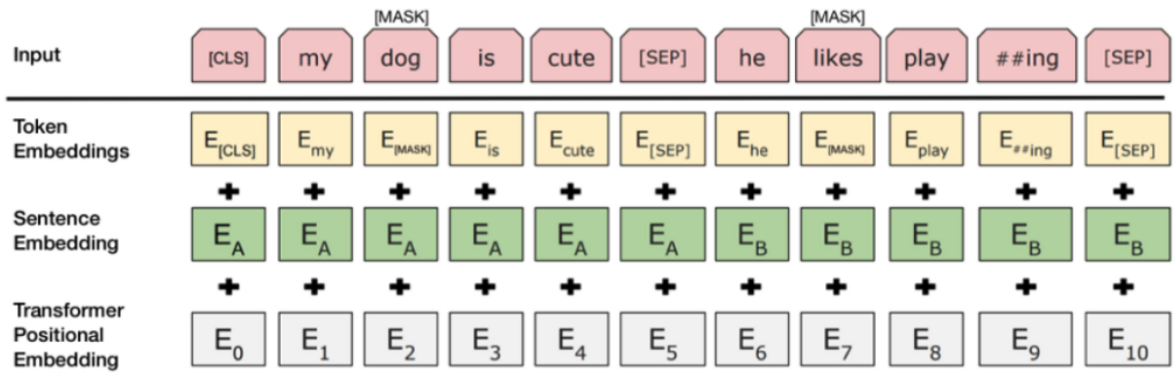


Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

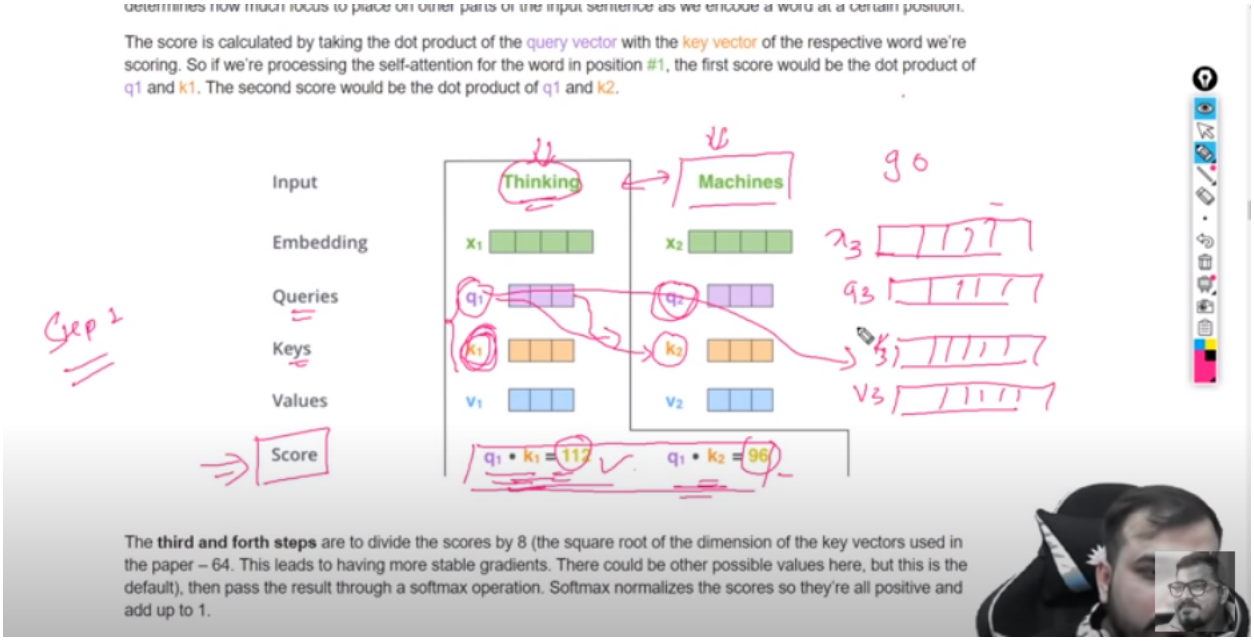
To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

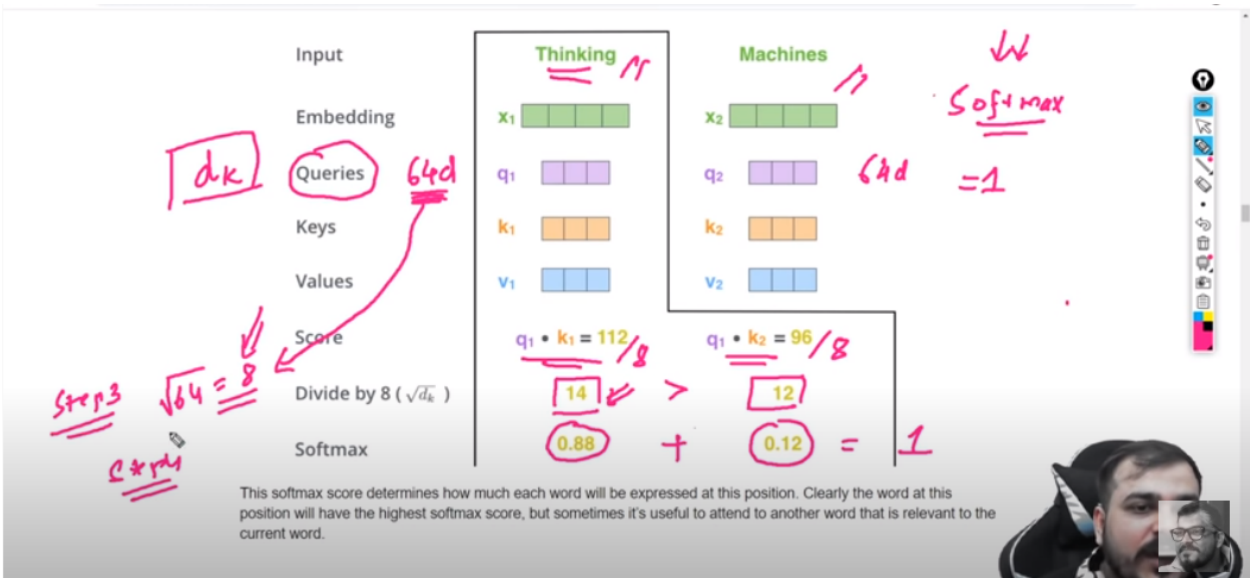
1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

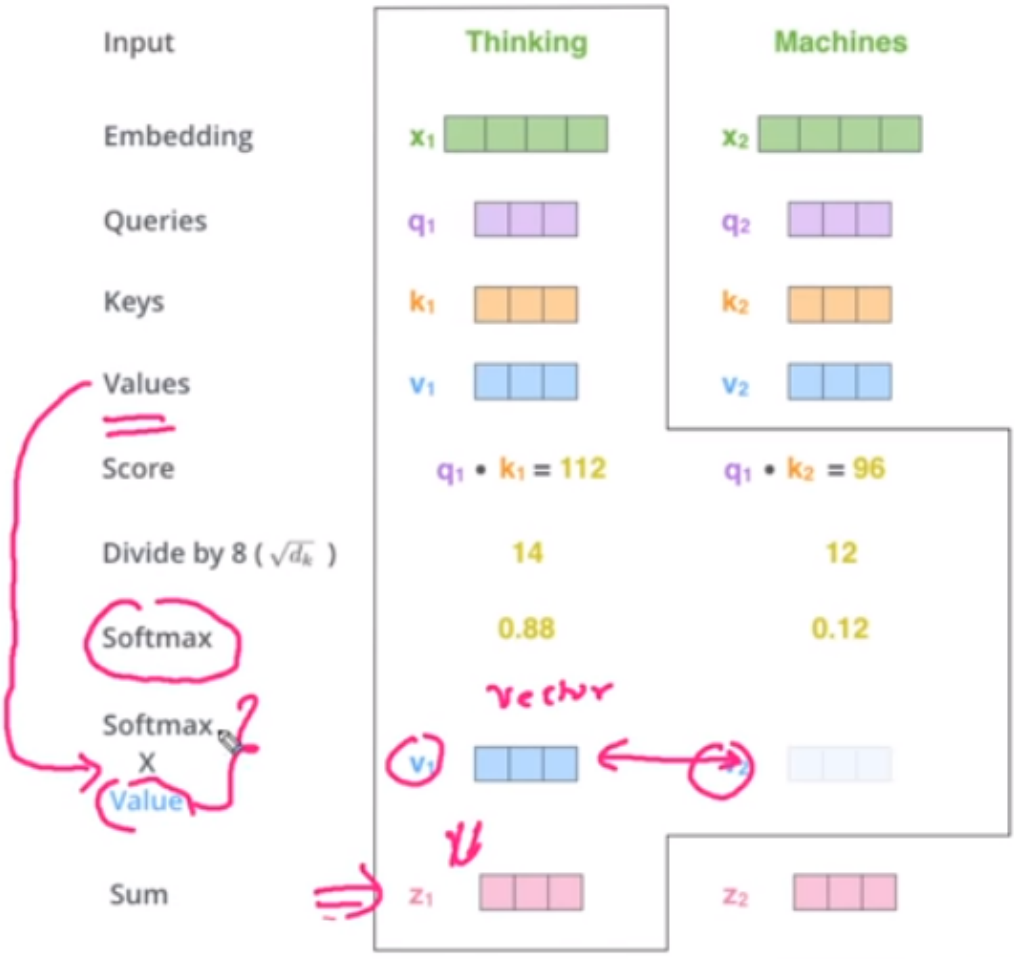


Source: BERT [Devlin et al., 2018], with modifications

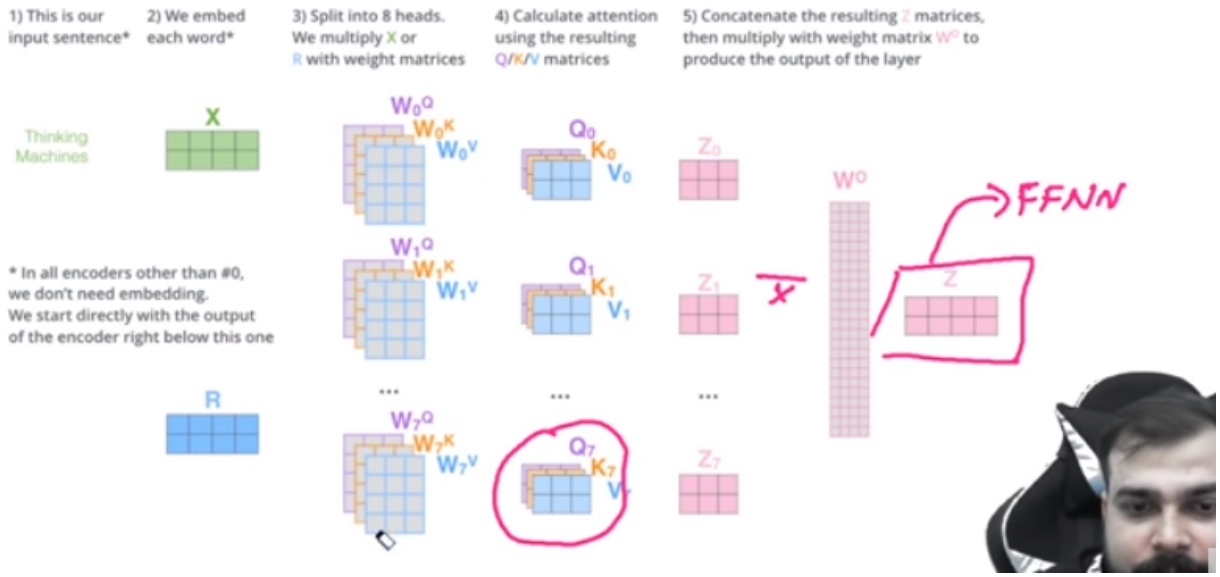
KRISH NAIK



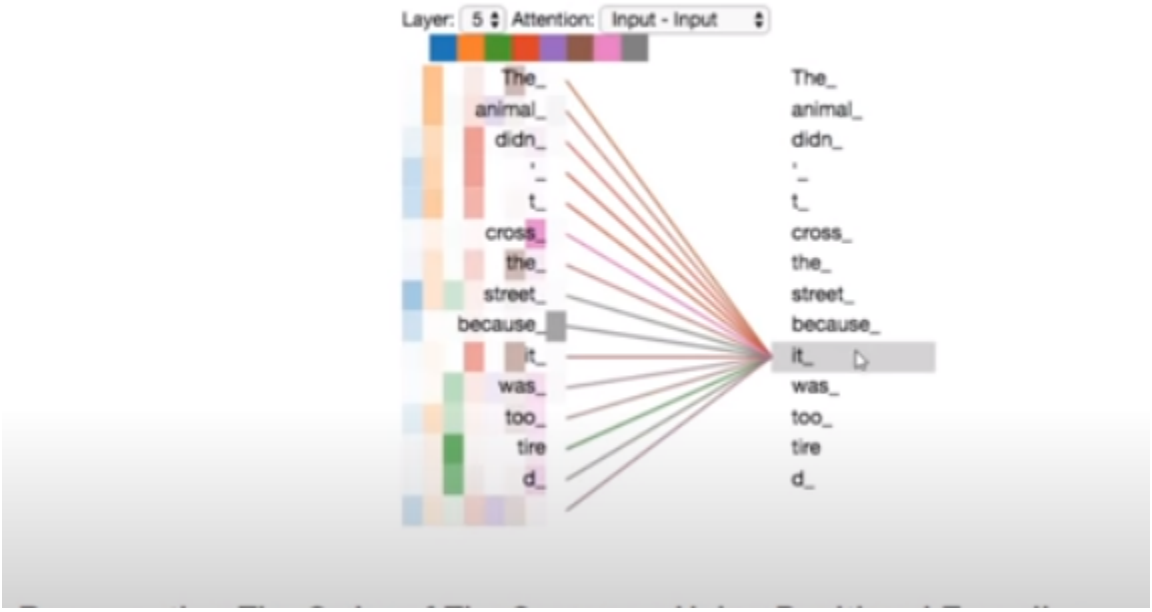

Input
Thinking
Machines
Embedding
Queries
Keys
Values
Score
Divide by 8 ($\sqrt{d_k}$)
Softmax
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
14
12
0.88
0.12
This softmax score determines how much each word will be expressed at this position. Clearly the word at this position will have the highest softmax score, but sometimes it's useful to attend to another word that is relevant to the current word.


Input
Thinking
Machines
Embedding
Queries
Keys
Values
Score
Divide by 8 ($\sqrt{d_k}$)
Softmax
Softmax X Value
Sum
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
14
12
0.88
0.12

That's pretty much all there is to multi-headed self-attention. It's quite a handful of matrices, I realize. Let me try to put them all in one visual so we can look at them in one place

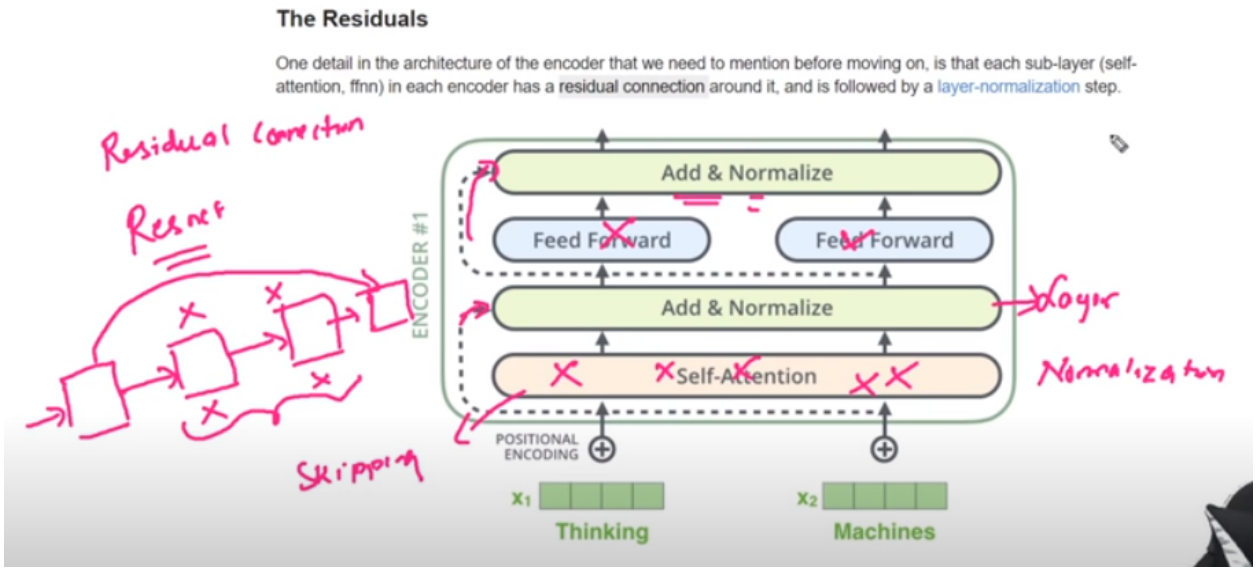


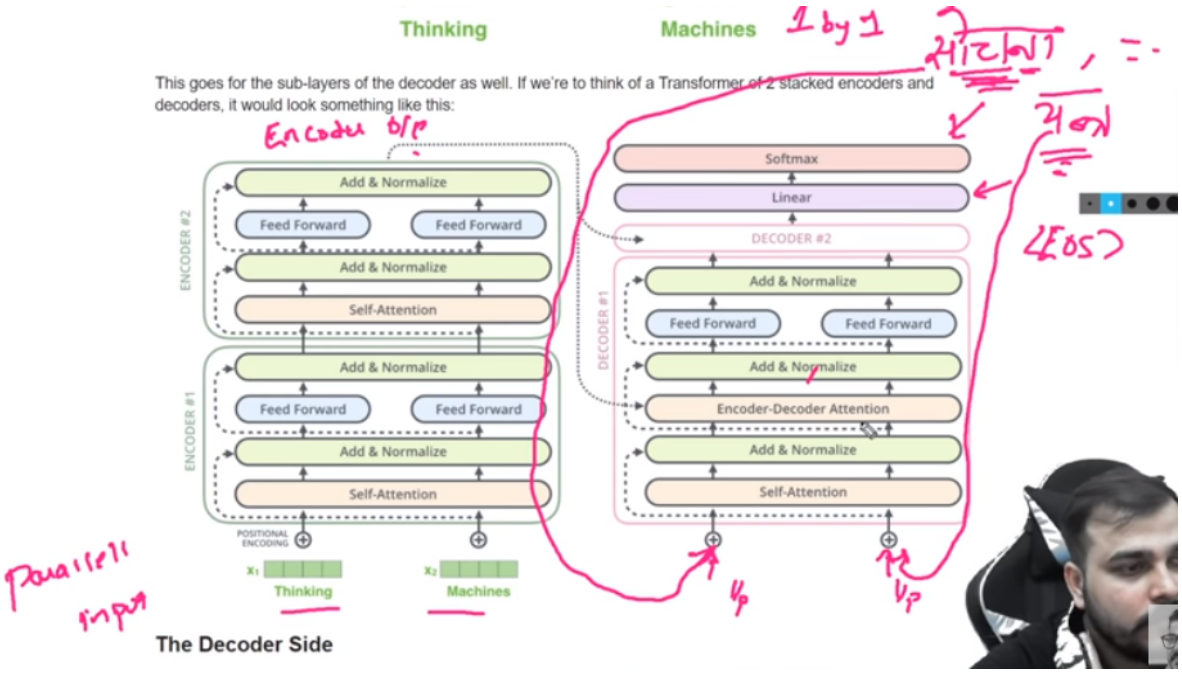
If we add all the attention heads to the picture, however, things can be harder to interpret:



4 positional encoding before passing to encoder.
Calculated based on distance.

5.





Decoder side

The output from the encoder is given to the decoder side as input. The decoder predicts some output. Now for the second word prediction by the decoder, the decoder output is again given to the decoder input along with encoder output directly to the encoder decoder attention layer of the decoder side.

add positional encoding to these decoder inputs to indicate the position of each word.