Kartik kiran karande

Karandekartik777@gmail.com

9527722700

Natural language processing using Transformers

In this blog we are going to see how Transformers changed the whole natural language processing , we will also look at what was the previous technologies and what was their drawbacks

The starting of NLP and Sequencing modelling starts with structure called RNN (Recurrent Neural Network) Then there are fewer advancement in the structure and then came LSTM ,GRU , Bidirectional RNN ,stack RNN then ENCODER-DECODER Then Transformers

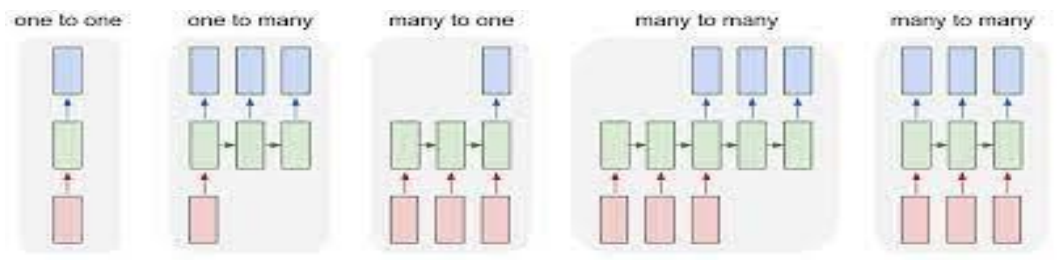
The paper Called “**Attention is all you need** ”came into picture and changed the whole Game. The author of this paper proposed a idea and Named it Transformer So now we will see why the transformer came into picture and how it outperformed all the NLP models

RNN: In Traditional neural network no previous information is saved for every new epoch or iteration. Every time new calculation take

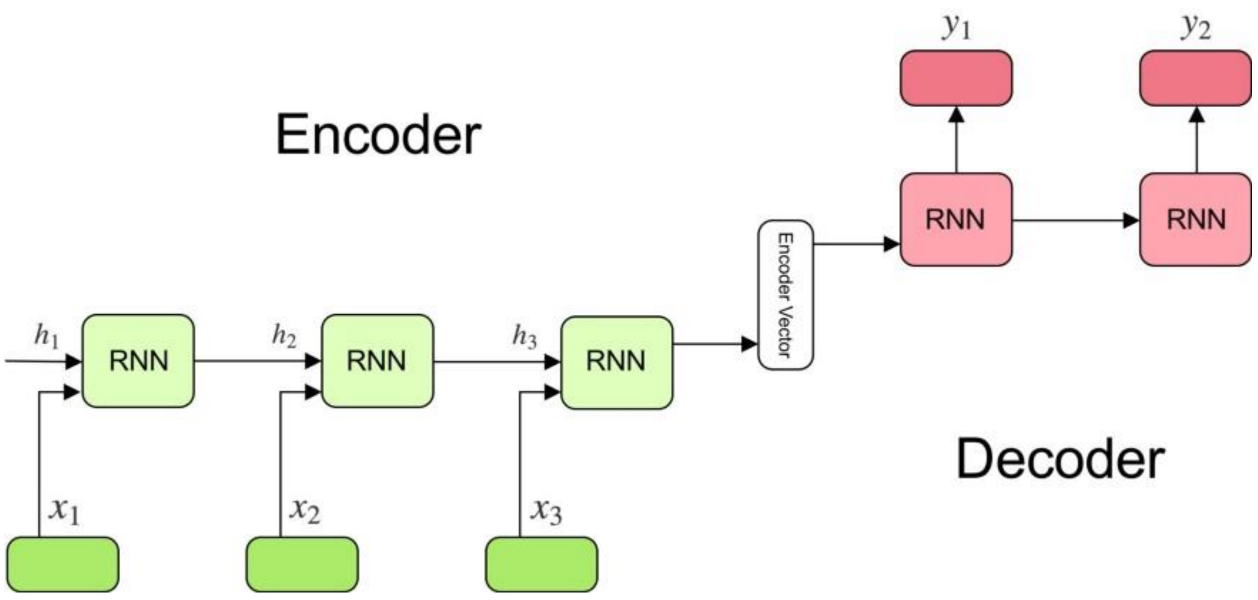
in RNN the output from the previous step is fed to the current step with its own input .

cases like predicting next word using the previous sequence is possible because of RNN thus RNN came into the pictures which remembers the sequence

But the RNN are not useful for long sentences so Bidirectional RNN came into existence ,But there is no contextual information sharing for long sentences so this models Failed



Encoder – Decoder The encoder-decoder architecture for recurrent neural networks is the standard Neural machine translation method that rivals and in some cases outperforms classical statistical machine translation methods



Transformers

To understand the transformers, you need some knowledge in RNN and Sequence models

In neural machine translation a sequence is play very important Role

The transformer model consists of stack of encoder and decoders

LETS UNDERSTAND TRANSFORMER

Encoder: encoder is the main and important part of the transformer there are stack of 6 encoders and each layer consist of many sub layers

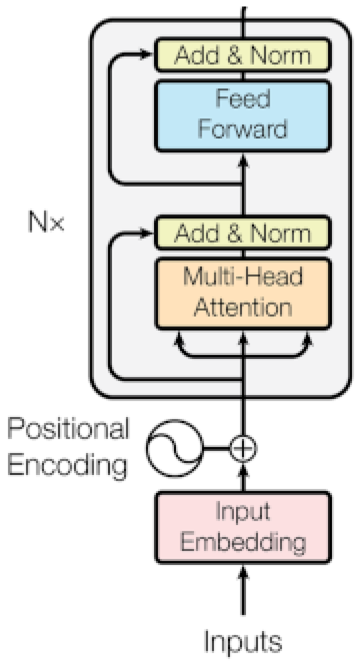
The sub layers consists of

1 multi headed attention

2 addition and normalization layers

3 feed forward neural network

Architecture of Encoder



The input embedding are nothing but the numerical vector representation of textual information of word or sentences the traditional approaches like word to vec can be used to generate Embeddings

Creation of Embeddings

ELMO : ELMO is nothing but Deep contextualized word representation

For better performance of any models the embeddings play very important role

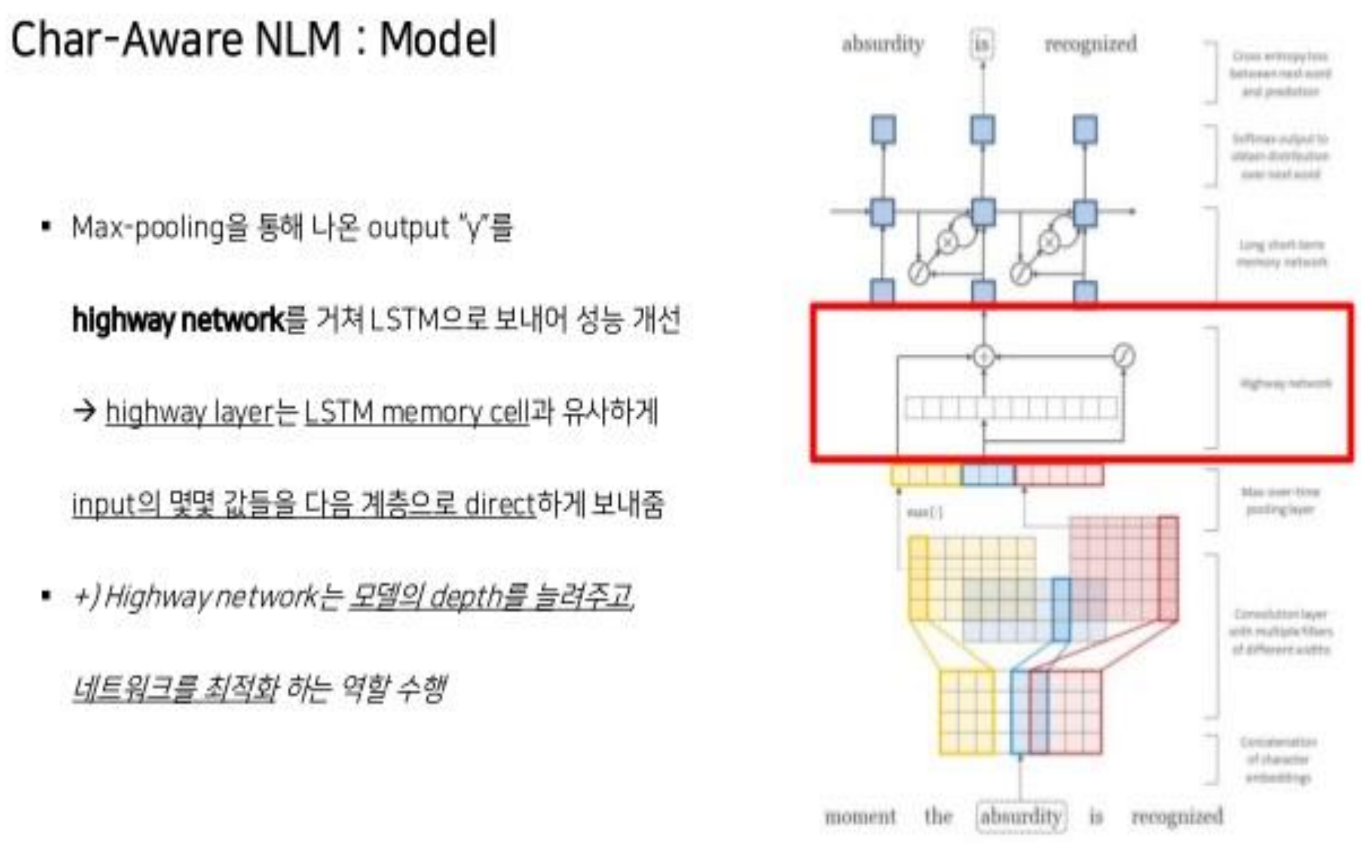
representations differ from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence. We use vectors derived from a bidirectional LSTM that is trained with a coupled language models and this language models are trained on very large amount of textual data corpus

The linear combination of stacked LSTM are used to create the Embeddings

After many trials and experiments its proved that ELMO works extremely well

The architecture of ELMO is derived from the paper called as Character **Aware Natural Languages Model**

ELMO ARCHITECTURE



Char-Aware NLM : Model

- Max-pooling을 통해 나온 output "y"를

  highway network를 거쳐 LSTM으로 보내어 성능 개선

  → highway layer는 LSTM memory cell과 유사하게

  input의 몇몇 값들을 다음 계층으로 direct하게 보내줌

- *+) Highway network는 모델의 depth를 늘려주고,*

  네트워크를 최적화 하는 역할 수행

absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
max{·}
Max over-time pooling layer
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

This ELMO architecture consist of

1.Concatenation of char embedding

 2. Convolution layers applied on char data embedding

3.max pooling layers

4. Highway network

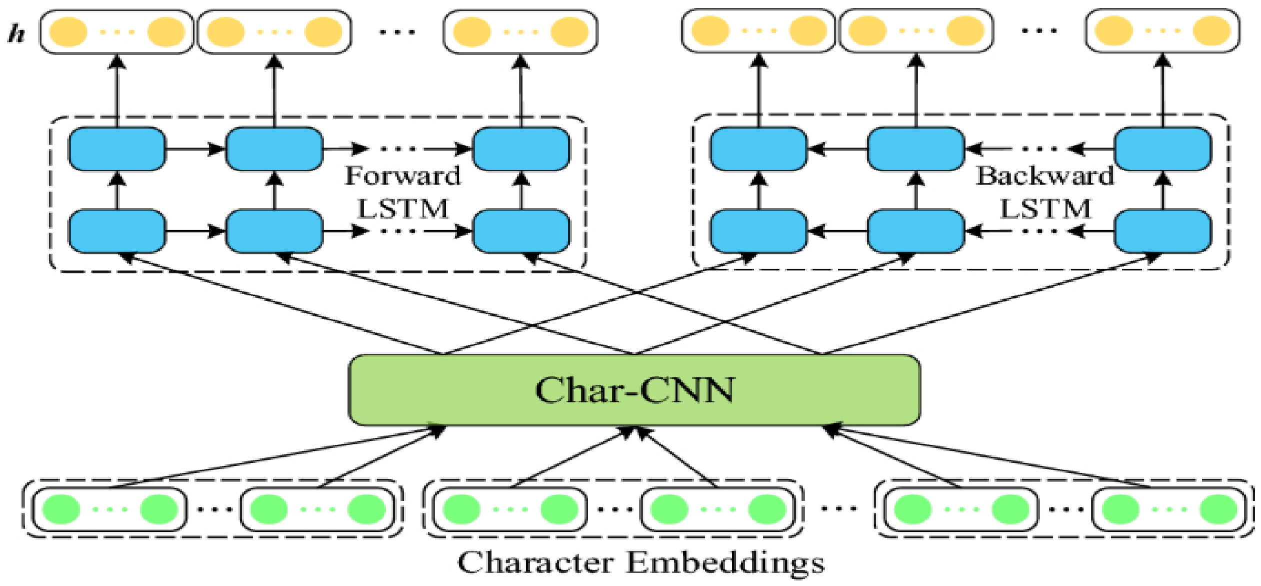
 5 Long layers of Bi LSTM

 6. Softmax output layer

 7.distance over next word

The text data is passed to the embedding layers the text in converted into numerical features then this feature are passed to the Convolution layers where the convolution operation takes place the Important features are taken into consideration using max pooling then the output of the Maxpool layer is passed to highway network which is a vector where all this feature are placed in single vector and operation of Bi directional **Lstm** is performed and this embeddings are ready to fed to any sequence generating model

SIMPLIED ELMO ARCHITECTURE



The output embedding of this architecture can be fed to models like and GPT and BERT

Positional encoding : it gives the information about placement of word in sentence . Position and order of words are the essential parts of any language. They define the grammar and thus the actual semantics of a sentence

Multi Headed Attention

Each sentence is converted into the embedding and then passed into the encoder the dimension of each word is 512 and this embedding are passed into self attention layers

Self attention :

Let me explain you what is exactly Self Attention is

Example

I am standing in front of bank and water is touching my feet

Consider the above sentence and try to understand the context of it

You will think one person is standing in front on a bank . for you the bank is a financial institute or anything

but if I remove the word **water** from the sentence it will hard to find the context.

which **Bank** I am talking about In this case **water** is **focus** word because due to this word we are able to understand context

For your better understanding consider

In preprocessing textual data weight is assigned to the each word and the word with higher weight is the focus word (attention)

SELF ATTENTION

SINGLE HEADED ATTENTION

Lets understand the self attention

For calculating the self attention we need to understand three things which are the

Key Vector ,Query vector, Value vector

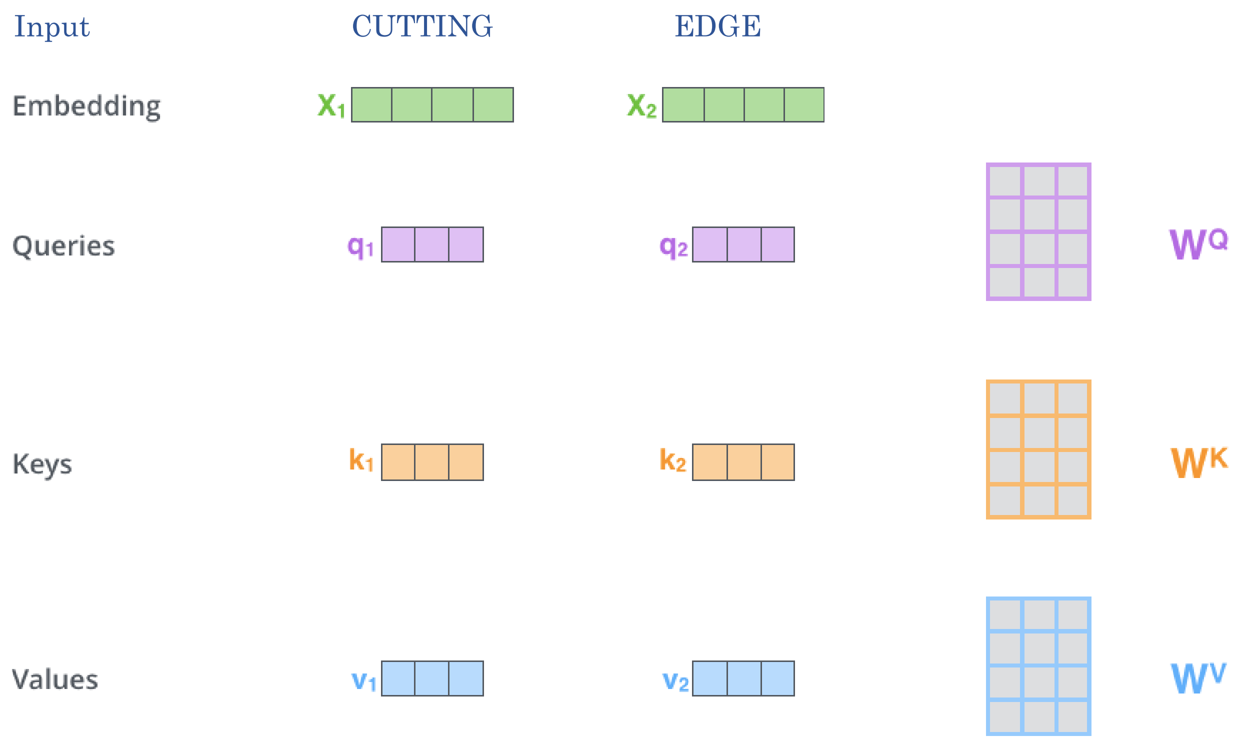
For calculation of output embedding this features are very important lets understand it

The key , Query and value are 64 Dimensional vector

Calculations

The values of key value and Query calculated as follow

Query = Q1 * Wq , Key = K1* Wk , Value = V1* Wv



So like this we calculate the values for each pair and decide which words context is important to prediction of upcoming word

After calculation of Key Value and Query we find the score for each word

After calculating score we divide each word by the square root of dimension of respective vector

That is

Score(cutting) = q1*k1 **Score(Edge) = q1*k2**

Cutting = 12/ square root(64) = 1.5 , **Edge = 24/square root(64) = 2.6**

After this step we apply softmax function on this values

Softmax(1.5) < softmax(2.6) i.e q1*k1 < q1k2

So here we can understand the the relation of word '**Cutting** ' with '**cutting** 'Is less than the relation of cutting with word Edge

means word 'cutting' is more important to predict the word 'edge'

Pseudo code to calculate Self Attention

words	cutting	Edge. ...... n	
query	$q_1 = \boxed{\ \ \ \ \ \ }$	$q_2 = \boxed{\ \ \ \ \ \ }$	W_Q $\boxed{\ \ \ }$
key	$k_1 = \boxed{\ \ \ \ \ \ }$	$k_2 = \boxed{\ \ \ \ \ \ }$	W_K $\boxed{\ \ \ }$
Value	$V_1 = \boxed{\ \ \ \ \ \ }$	$V_2 = \boxed{\ \ \ \ \ \ }$	W_V $\boxed{\ \ \ }$

Score calculation

① Score (cutting) = $q_1 \cdot k_1$ Score (Edge) = $q_1 \cdot k_2$

② divide score by the dimension of key.

① assume $q_1 \cdot k_1 = 12$ and $q_1 \cdot k_2 = 24$

$S_1 = \frac{\text{score}}{\sqrt{\text{Dimension of key}}} = \frac{12}{\sqrt{64}} = 1.5$, $S_2 = \frac{\text{score}}{\sqrt{\text{dimesion of key}}} = \frac{24}{\sqrt{64}} = 2.6$

③ Apply softmaxe on Values of S_1 and S_2 .... n

$$\frac{e^{1.5}}{e^{1.5}+e^{2.6}} < \frac{e^{2.6}}{e^{1.5}+e^{2.6}}.$$

④ Multiply Vector of softmaxe with Value Vector

$$\sum_{i=0}^{n} \text{softmaxe} * \text{Values} = z \text{ (attention function)}$$

$$Z = \left(\text{softmaxe}\left(\frac{q_i \cdot k_i^T}{\sqrt{d_k}}\right) \cdot \text{Values} \right)$$

Kartik Karande

then we multiply the score value vector with the softmax and after this multiplicatioin we take summation over all the Value’s

Multi-Headed Attention

in multiple attention there are **stack** of **Query , keys** and **values** .

These multiple head attention is used **to improve the performance of attention layer**

the increased attention heads are **useful to concentrate on different position in sentence** or text

we will get multiple representation subspace using multiple attention heads

The multiple heads are useful to understand the Relationship

Multiple heads are used to Generate understanding of punctuation, Textual Relationship some are used to understand the grammar

some are used to understand positional Information, context, word arrangement

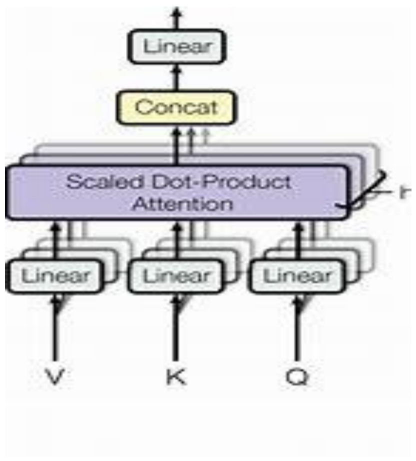
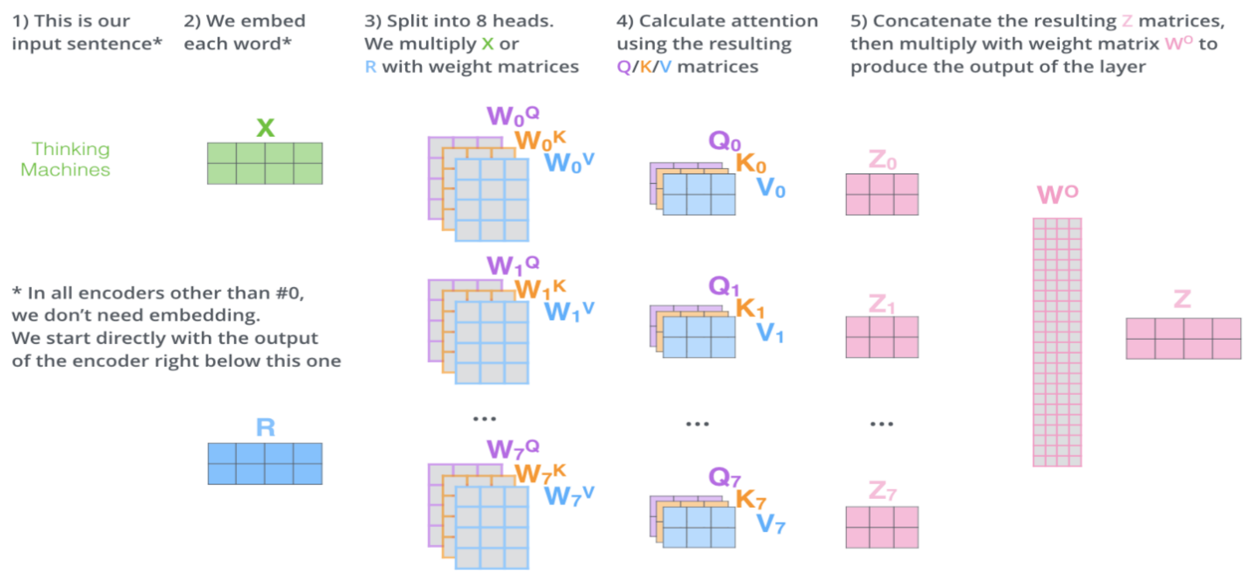


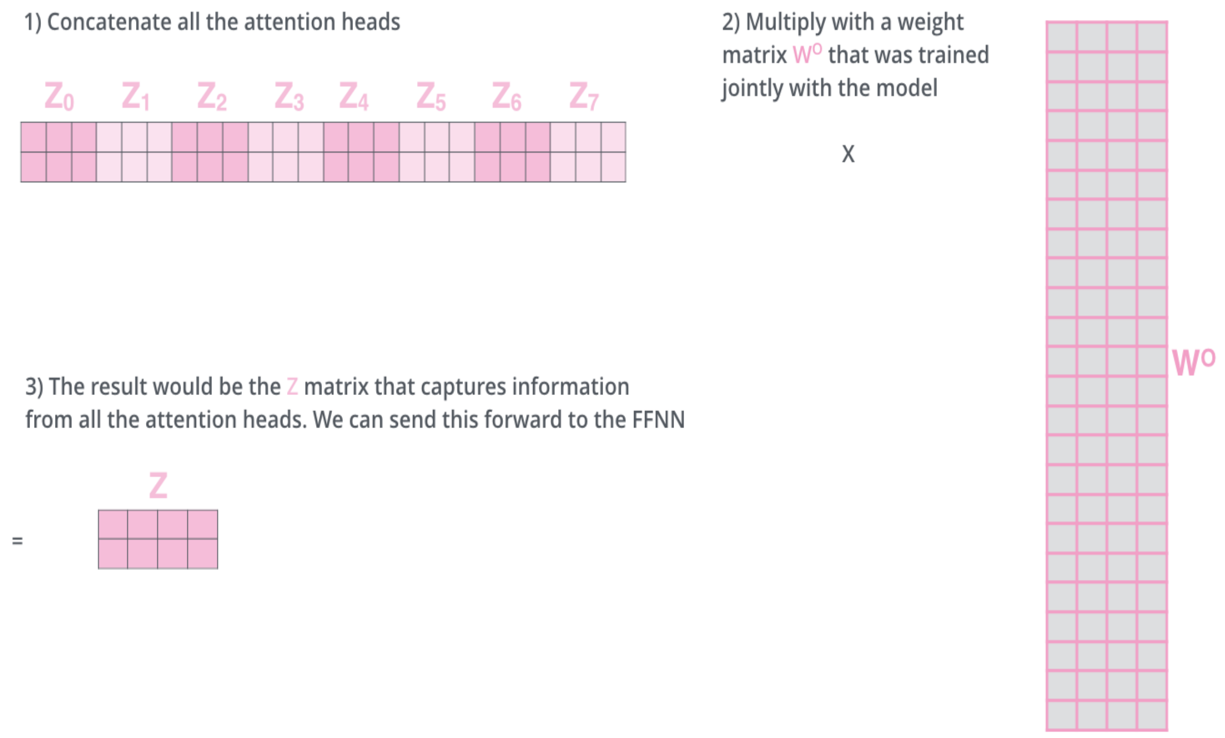
figure of stack of Multi head attention


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z

How final attention vector is calculated :

To calculate the final attention vector We stack all the attention heads and multiply then with Wo matrix and finally We get a matrix **Z** which consist the information of all the attention all attention blocks

Figure below shows how the final attention vector is created from stacked weights


1) Concatenate all the attention heads
Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z

The Final attention Block **Z** is passed to the addition and normalization layer of encoder

Inside the addition and Normalization layer the Layer normalization is applied on the Z vector

The out from Addition and Normalization Layer then passed tto FORWARD NEURAL NETWORK this Neural network consist of Residual Dropout so the maximum Required information is kept

The output from the Feed Forward Neural Network is again passed to the addition and normalization layer

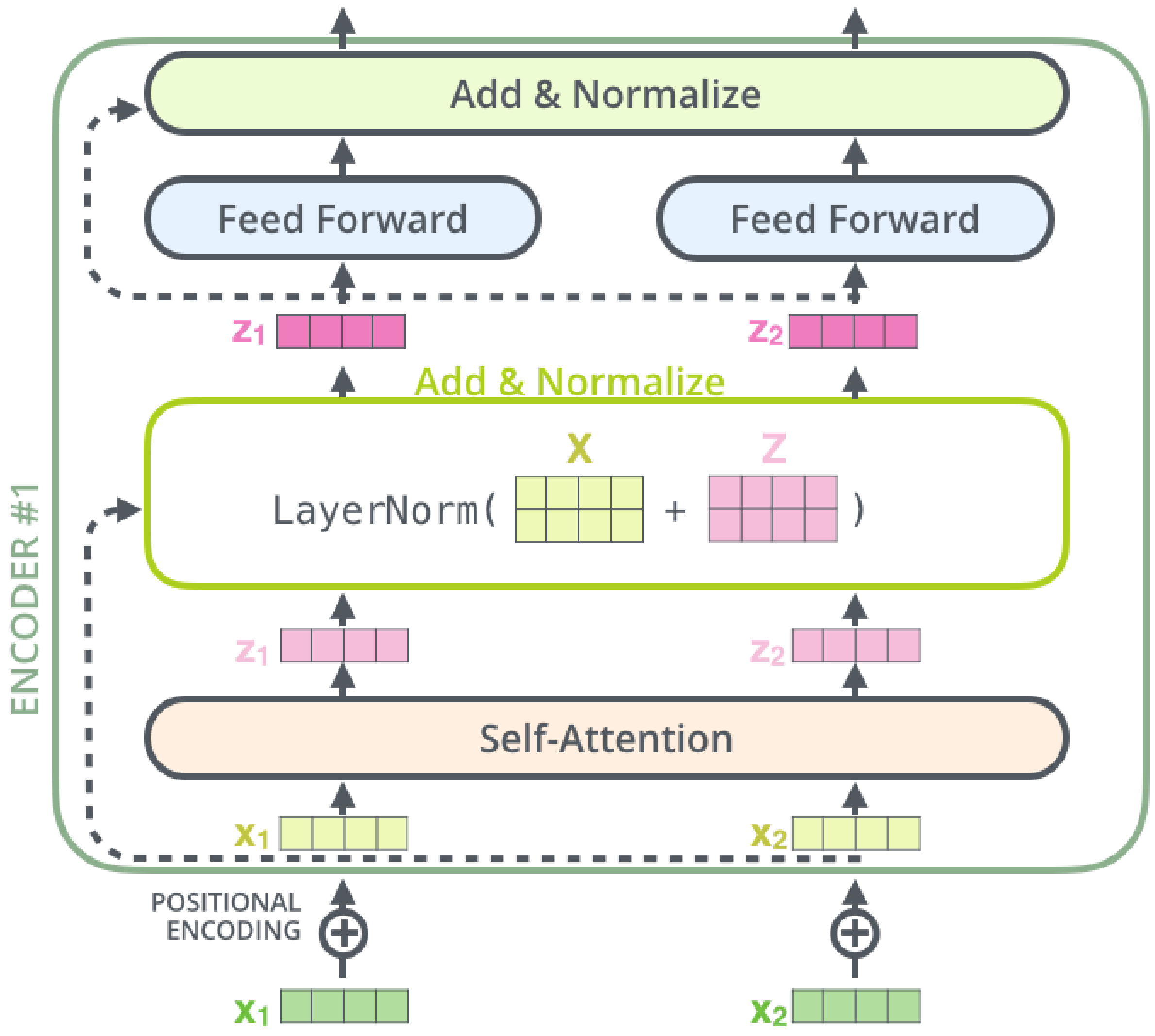
In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

FFN(x) = max(0, xW1 + b1)W2 + b2 (2)

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is = 512, and the inner-layer has dimensionality df f = 2048.

This how computation in one Encoder Block is Done and there are such 6 encoder blocks

The Output From final 6th Encoder is Passed to the Decoder Network to calculate the output



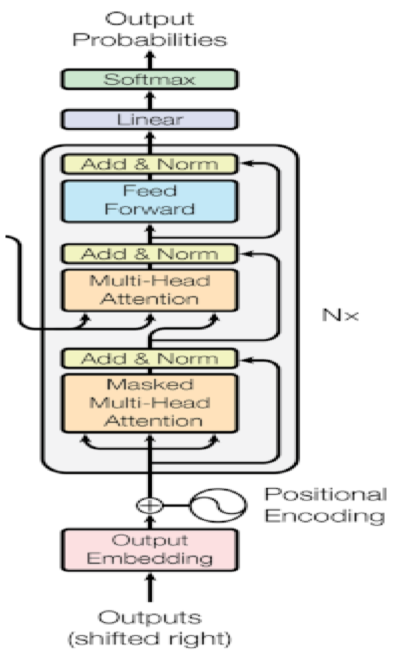
This is a figure of single Encoder Block inside the Transformer

THE DECODER

The Decoder block is responsible for generation of output

Output from the last layer of encoder layer is passed inside **Encoder-Decoder attention layer** of Decoder block as well as the **Self attention layer** of Decoder

The output which we get from the **Encoder-Decoder attention layer** of Decoder is Fed to **Feed forward Neural Networ**k and output from the Neural network again passed **to Addition and Normalization layer**



The Real Architecture shown in Research Paper

This is how calculation in One Decoder block is performed and there are such 6 Decoder blocks stacked over one another

DECODER ARCHITECTURE

Each Block of Decoder is Consist Of

Self-attention Layer

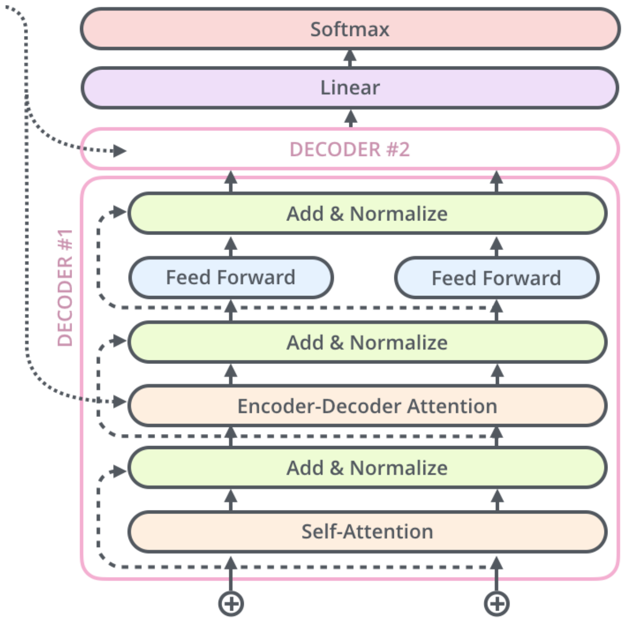
Addition and Normalization layer

Encoder -Decoder Attention Layer

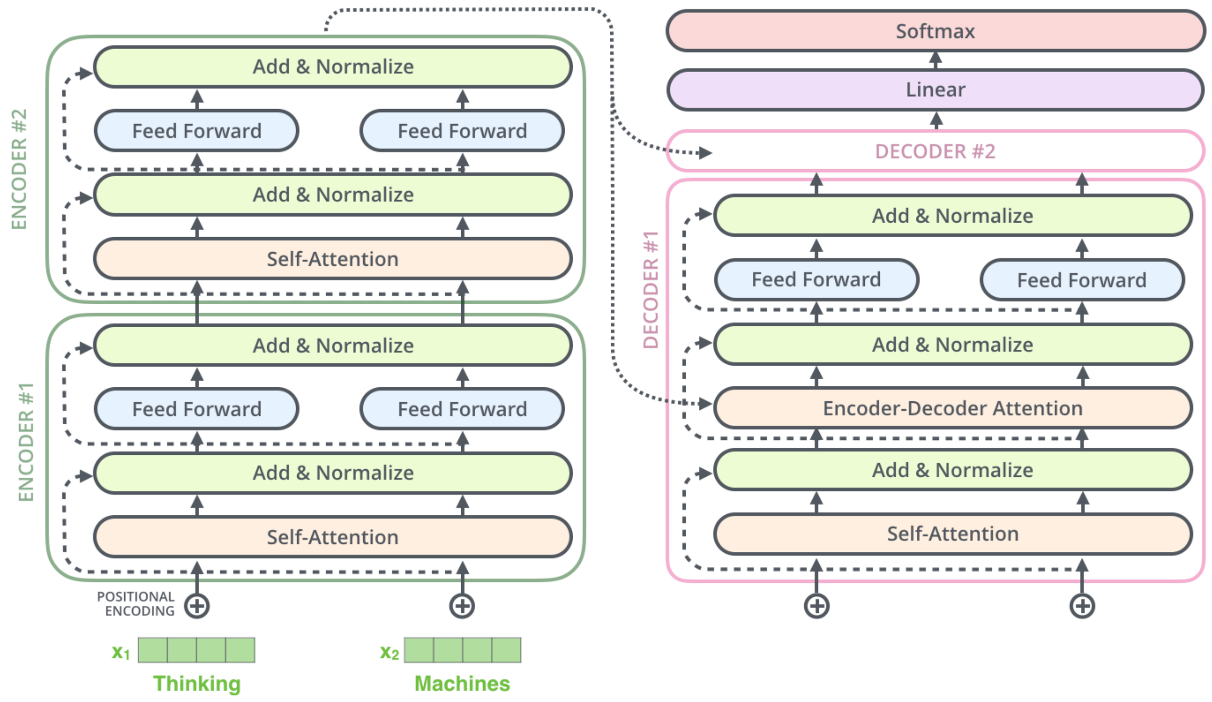
Addition and Normalization Layer

Feed Forward Neural Network

Addition and Normalization layer



Simplified Decoder Architecture



Simplified Architecture of Encoder -Decoder

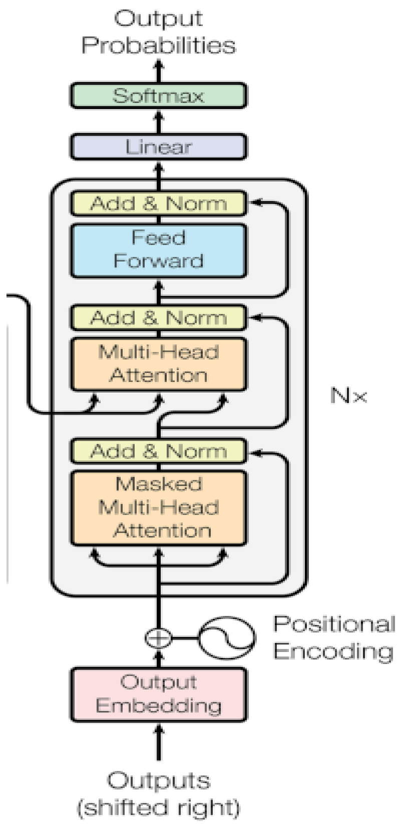
MODEL TRAINING

This attention model is Tested on **English and German** translation dataset this data consists **of 4.5 million sentence pair** this sentence was encoded **byte by byte pair Encoding** and the vocabulary consist of unique **37000** tokens

This model is also trained on **English French dataset** which consists of **36 million** sentence pairs

Each training batch contained 25000 source and 25000 target Tokens

Adam Optimizer and Residual Dropout of 0.1 is used for Regularization



Positional Encoding: The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence and these positional encodings are play important role in word positioning in sentence to create a context

Masked Multi-Headed Attention : in masked multi headed Attention we hide some word inside the sentence and our models try to predict the word

Example 1.

Masked sentence

<SOS> You only **[Mask]** once, but if you do it right, once is enough."<EOS>

<SOS> You only **LIVE once**, but if you do it right, once is enough."<EOS>

The machine tries to predict the word which is masked

In Case of language translation

English sentence : In nature, nothing is perfect and everything is perfect.

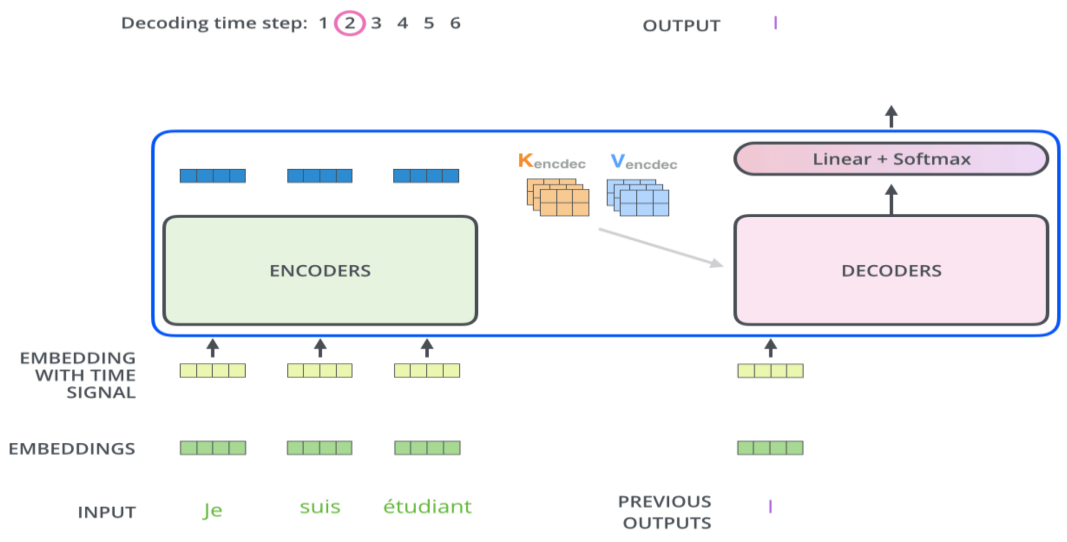
GERMAN Translation: In das Natur ist nichts perfekt und alles ist perfekt.

In this translation "**der"** is expected but model predict "**das"** so we can correct this using backpropagation and weight updation

LOSS CALCULATION :

Inside the loss calculation the Neural network tries the correct the word which is false predicted using backpropagation

Lets understand this using the animation



LAST DECODER AND SOFTMAX LAYER :

ABOVE THE LAST DECODER GIVES THE OUTPUT AS A FLOAT VALUE VECTOR WHICH IS CONVERTED INTO WORD USING LINEAR AND SOFTMAX LAYER

LINEAR LAYER :LINEAR LAYER IS A FULLY CONNECTED NEURAL NETWORK WHICH CONVERT THE OUTPUT VECTORS GIVEN BY DECODER TO LONG VECTOR

SOFTMAX LAYER:- SOFTMAX CONVERT WORD PROBABLITY VECTORS INTO THE WORDS THE HIGHEST PROBABILITY VALUE IS CHOOSE TO GENERATE THE WORDS

State of the Art Model

Base of All state of the art NLP models is transformer

Models like BERT , GPT used transformers architecture as base model

BERT

Bidirectional Encoder Representations From Transformers

Bert is the State-of-the-art NLP model created by Google . **B**ert uses Bidirectional training for the transformer it is accurate at less computationally expensive , It takes text sequencing in understanding from both side (left to right) &(right to left) there for it is easy to understand the context of **[Mask]** word

Bert models can be also used for **Next word prediction , Name entity Recognition, Question Answering**

While training the few words in sentences are masked and Model try to predict the word and if predicted word is wrong then we calculate the loss function and By updating weight correct word is generated

Architecture of BERT Model

The BERT Model used only encoder as its architecture on the top of last encoder layer there is a Feed forward Neural Network which is used for training

Above that FFNN there is Linear and softmax layer for output production

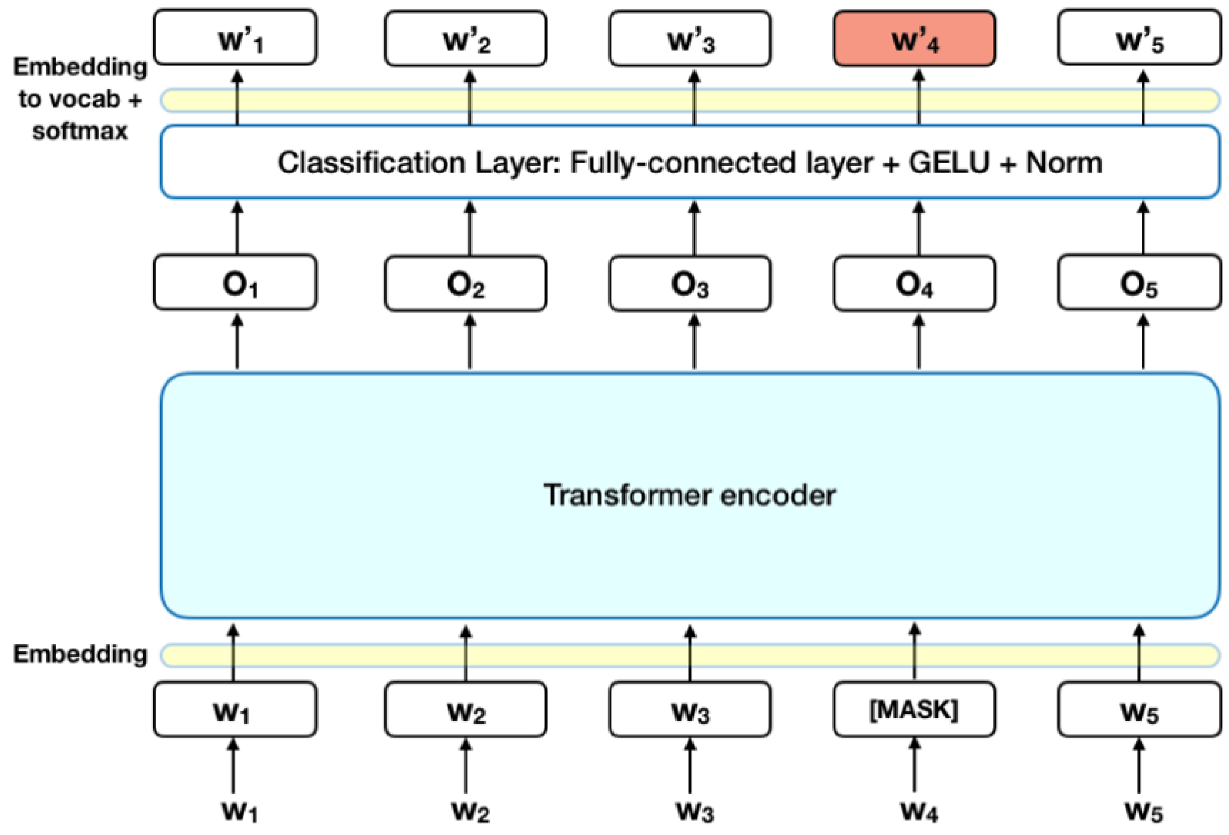
Lets understand how BERT understand the masked word

Cat is **[Masked]** on the table

Inside BERT masked word is predicted using the previous word **'cat is'** and next word **'on the table'** .

How much previous and next word use for prediction is a hyperparameter

But due to Bidirectional Nature Word prediction is easy and only because of Bidirectional Nature it perform Better than openAI's GPT



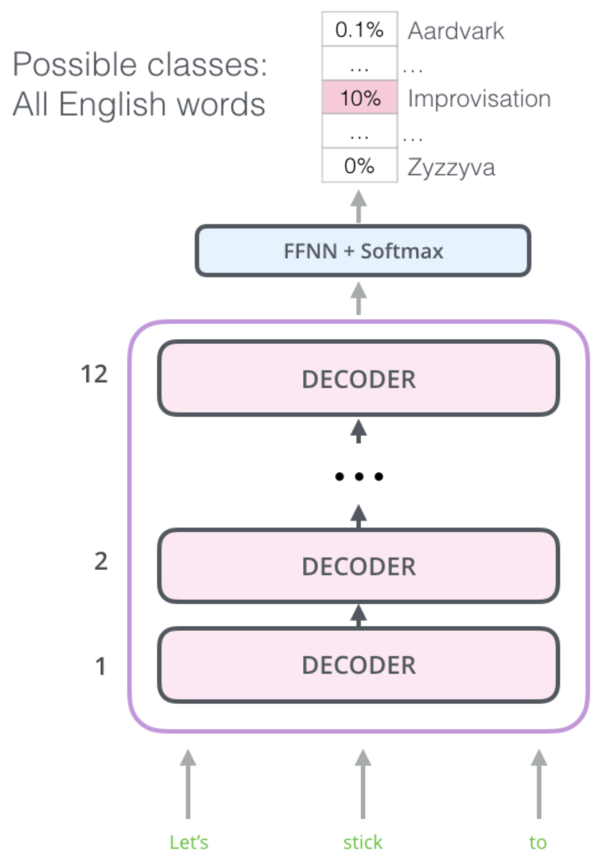
GPT

GPT is another state-of-the-art model Created by open AI

GPT used DECODER Network as base model

The GPT Model used only Decoder as its BASE architecture on the top of last Decoder layer there is a Feed forward Neural Network which is used for training and Linear and SoftMax layer for converting Numerical vector values to word

GPT uses unidirectional approach for next word prediction and that's why it fail sometime



GPT ARCHITECTURE

I Hope this Blog gave you understanding of how TRANSFORMERS works and How it played important Role in generation of STATE OF THR ART MODELS

THANK YOU…