Transformer 101.

↓

Power behind the modern day AI boom, powers tools like ChatGpT.

Initially we see language Models (Machine learning models like Bird that basically predict the next word in our text based on analysis of the previous words. from Google)

e.g (in emails) → writing a half way through msg is usually predict to be completed pressing [tab]

I hope my email finds you well .

↳ dotted line shows the predicted words the language model.

* Transformers are doing just that!

basically are Large language Model (Deep learning trained on more dataset) that catches a key word from text and predicts, and predicts and predicts ... until it generates a complete answer to our prompt.

UNDERSTANDING WORD EMBEDDINGS.

ML/DL model only understand numbers, thus we need to represent words into numbers. but how?

↳ By asking bunch of questions.

Imagine Representing the word King

male =?
gender =?
coward =?
Honest =?

$$\begin{bmatrix} 1 \\ 1 \\ 0.2 \\ 0.5 \end{bmatrix}$$ → makes up a vector

Now these bunch of questions are really not wats happening rather instead there are features of the vector for king that has a particular dimension and usually perfected through backpropogation.

Models like GPT has 12000 dimensional vectors.

Static Embedding.

Word2Vec & GloVe are two models that can generate fixed/static embeddings.

↳ These are generated by training a neural Network model on large amount of text to understand the words; thus a vector with features.

Now static embeddings for a word can be a problem as the the meaning of a word changes from context to context of a sentence.

eg I made a rich dish.
I cleaned the dish washer

} for both we may need embeddings slightly different from one another as per the word's context in the sentence.

this heavily effects correct predictive mechanism. ←

because.

I made a rice dish called → risotto
→ idli
→ mexican rice

OR.

I made an indian rice dish called → idli
→ Biryani
→ kheer

(dotted underlines represents the given prediction need to be expected)

This issue is solved by the 'conceptual Embedding'.

↳ take the original static embedding of the word and let it influence it/change it to eventually change the n-dimensional vector.

Transformer Architecture.

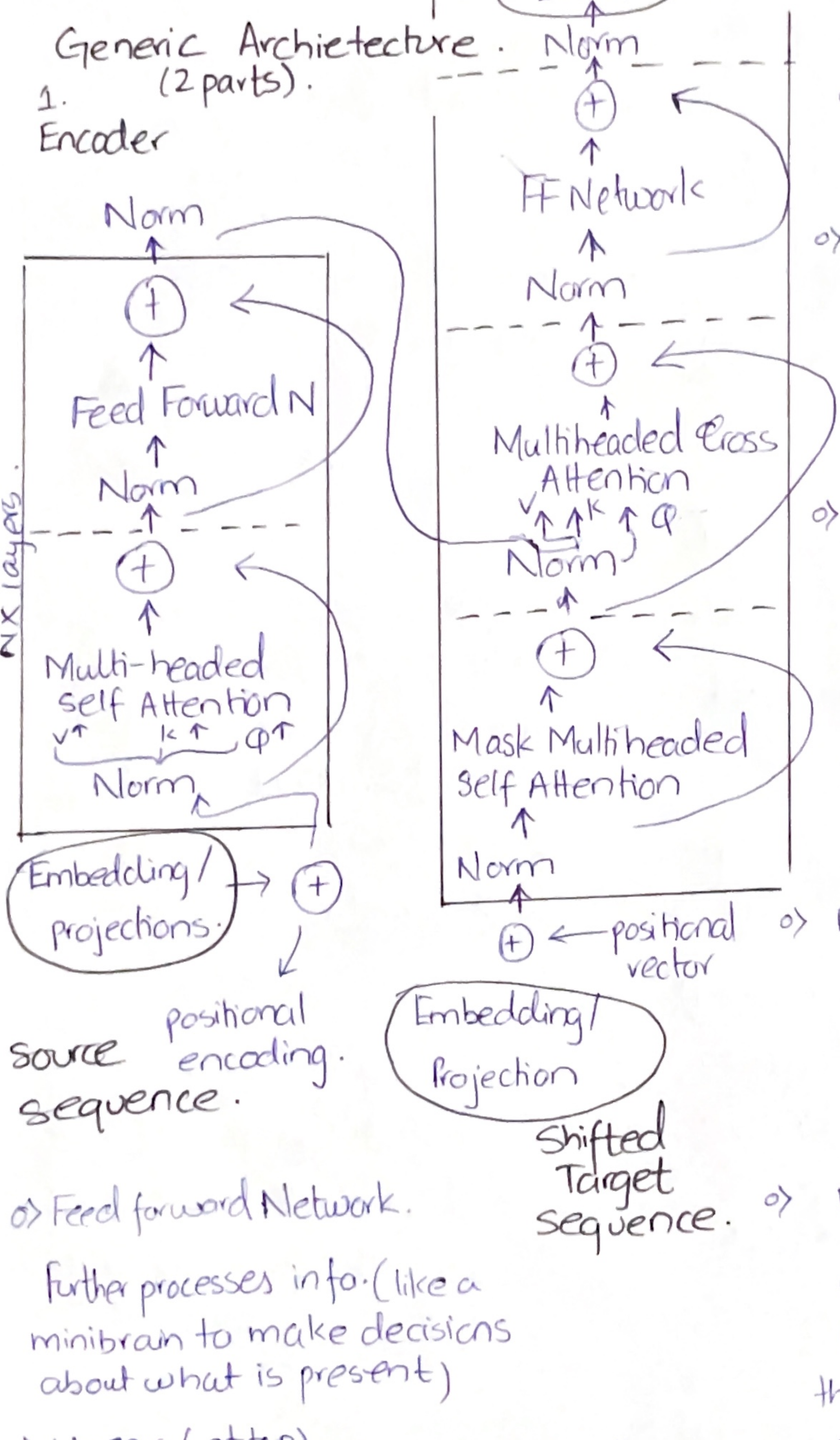
→ encoder takes in sentence and generates a conceptual embedding for each word in the sentence

Deep learning models have two stages.

(i) Training.
(ii) Inference.

Generic Architecture. (2 parts).

1. Encoder



- source sequence: input text
- embeddings/ projections: This converts word into no.s (vectors) that the computer can understand.
- positional encoding: Adds information about where each word appears in the sentence; helps the model note/ understand word order.
- Norm: (Normalization layer) keeps no. in a manageable layer, helps the model to train more stably.
- Feed forward Network. Further processes info. (like a minibrain to make decisions about what is present)
- Linear (at top) final layer to convert processed info into actual prediction (predict words)
- Multiheaded self Attention: allows each 'word' to basically pay attention to other words thus focusing on d/f types of relationships simultaneously.
- Multiheaded Cross Attention. similar to self attention but connects words b/w input and output sequences.

COMMON TRANSFORMER ARCHITECTURE

BERT Architecture.

Contains only the encoder part i.e generates only the conceptual embeddings.

GpT Architecture.

Contains only the decoder part

The conceptual embedding is with in the decoder part.

* Based on the model/Architecture selected the dimensions of the embeddings generated vary.

- BERT (Bidirectional Encoder Representation from Transformers); developed by Google

- It is a Bidirectional meaning it looks at words before and after the target word to understand context.

- It is pretained using MLM and NSP

(Masked Language Model)

MLM: Randomly masks words in a sentence and tries to predict them using surrounding context.

NSP: Determines whether one sentence follows another in a document.

(Next sentence Prediction)

- Generative Pre-trained Transformer.
(developed by Open AI)

auto regressive text Generation

Using the decoder part of the transformer, uni-directional meaning it generates text word by word considering only past words.

Trained using CLM

(Causal Language Model).

↳ predicts the next word given the previous word in the sequence.

Tokenization Positional Embedding:-

Now further analyzing the process of Transformer working, we look how the embedding is carried on. (text based Model as an example:

1. Input the sentence sequence.
2. Tokenization (based on a specified method).

e.g I made a sweet Indian rice dish called

BERT tokenization.

3. These tokens are converted into Token ID which are then marked/cross checked from the static embedding matrix generated by the NN training process

4. This finally gives a n-dimensional static embedding/vector for that particular word which is now understandable by our deep learning model.

* Now remember transformer will process these words(now embeddings) simultaneously in parallel — (unlike RNNs) for this we need to keep track of the orders of our words.

↳ for this we use the special "Positional Embedding Technique".

5. adds a small vector into each static embedding (of each word) generated. ← ~ the resultant vector will embed in itself the knowledge of position of each word.

formula for Positional Embeddings

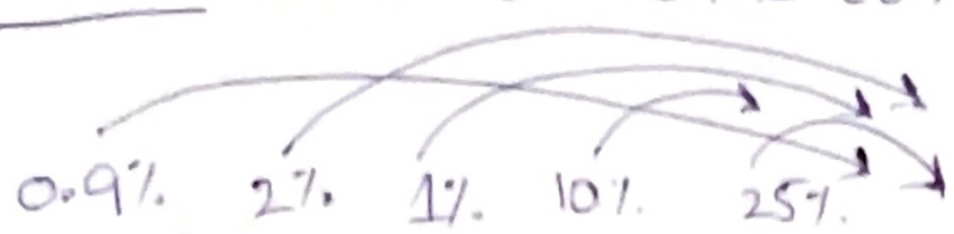
$$PE(pos, 2i) = \sin(pos / 10000^{2i/d_{model}})$$

$$PE(pos, 2i+1) \cos(pos / 10000^{2i/d_{model}})$$

ATTENTION IS ALL YOU NEED

2017 Research.
↳ By Googlers.

Now this research gave the idea how different words in a sentences are attending/re-enforcing a main word which is the center of attention (and needs that attention!)



e.g I made a rich Indian dish called .... this all helps us to derive contextual meaning.

↓
main word

Understanding Query, key, Value. (these words help us achieve the attention mechanism) through analogy

①	looking for a book on Quantum physic, Quantum computing etc.	Book rack, Book description Book index	Actual content of the book.
	⇝ What your searching for?	⇝ The labels used to identify relevant information as per the query.	⇝ The actual content.

In transformer: **Words interacts with each other using these 3 components**

- Each word in a sentence acts as a query, searching for relevant context.
- Other words acts as keys, indicating what they contribute.
- The words that contribute information are called/ provide their values.

"The sweet Indian dish was delicious"

Consider Sentence Processing example.

- The word 'dish' (Query) seeks context of the other words.
- words like 'sweet', 'Indian' (keys) describe it. (etc.)
- The contributions from these words (values) enriches the representation of dishes.

↓ actually vectors

* all of the value's vectors/static embeddings for (a) say (a) single word are added together to produce a context-aware embedding

" Now Dot product of Query and key will measure relevance/attention score (higher score ⇒ stronger attention)

→ SIDE NOTE

HOW TO GENERATE ATTENTION SCORES OR. HOW ARE VALUE'S VECTORS ARE BUILT :- (ATTENTION HEAD MECHANSM)

① get the static/positional embeddings of each word in the sentence.

② Select One word for/as a query and multiply its static embedding with a W_q matrix (whose dimension depend upon the transformer Model).

③ Resultant Query vector for that particular word selected

④ Now take positional/static embeddings of all the tokens/word and multiply them with another matrix i.e W_k to get key vectors.

[Repeat this process (1–3) for each word.]

⑤ Multiply the Query vectors and key vectors together to get a numerical values for each.

* ⑥ These values are passed through a softmax function to convert the numerical values into probability dist= Attention scores

⑦ Now for the Value's Vectors we will again take the static/positional embeddings and multiply it with another matrix i.e Wv

HOW TO GET CONTEXT AWARE EMBEDDING

⑧ Multiply the resultant of each with the attention scores generated for each (given/drawn previously); sum all the resultant vectors to finally get the context aware embedding (vector).

(after following all the steps previously assigned nowif)

*Remember this whole process is repreat for each word chosen as a query to get its context aware embedding ——— (a)

Till Now yove done.

flowchart Basic_1

Now keeping point (a) in mind;
flowchart Basic_1 is basically runned for many other group of words such as.

① generating context aware embedding for adj,
② " " " " for pronouns and so on.
→ linear transformations.

That is why the Multi Attention heads Transformer Model showcases
→ this mechanism allows model to focus on d/f types/aspects of relationships b/w tokens;

Residual Connections

All across the transformer models we see some arrows b/w the early and the later part of the module, called as the Residual connections.
- → maintains integrity of original input features.
- → allows the network to be more deeper.
- → helps in gradient flow during back prop.

They solve the core problem of "vanishing gradients".

Note:

They/Residual connections create a highway for information. They allow the original input to flow to the later layers.

The formula is simple Output = $F(x) + x$.

$F(x)$ ↓ transformed data

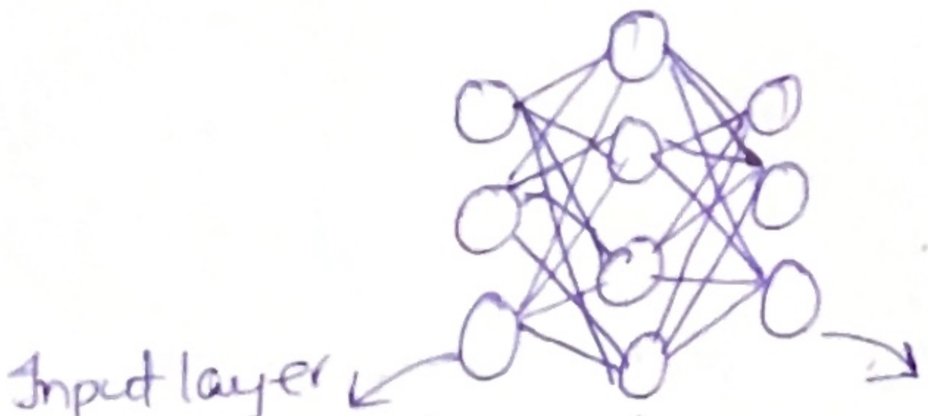
x ↘ original data.

↝ Without them deep networks face information loss, as data passes through many layers (important details can get losed).

↝ Training difficulty increases without them as very deep networks become hard to train effectively.

Feed Forward Network:-

After the generation of complete context Aware embedding by the (Multi head Attention layer) we pass it through a layer of fully connected Neurons i.e Feed Forward Network



⇒ Resultant :- enrich each token's embeddings by applying non-linear transformation independantly enabling the modle to capture more complex patterns and higher order features beyond contextual background.

* the weight and parameters are adjusted during backprop.

* Normalization ensures stable learning improving gradient flow.

* Nx layers ⇒ represents transformer block (how much they are being repeated).

e.g BERT Base: 12 layers.
BERT Large: 24 layers.

Output of Encoder ⇒ Contextual Rich Embeddings ⇒ Input to the Decoder.

MULTI HEAD CROSS ATTENTION

self attention: Words look at other words with in the same sequence.
↳ cross attention: Words look at words from the other sequence.

Basic functions:

↳ Connects encoder (input) and decoder (output).
↳ helps decoder figure out which part of the input to focus on when generating each output word.

Uses three components.

- Query: Comes from the decoder.
- Key and value: Comes from the encoder.

Example: 'English to spanish' translation

Input: The black cat

When generating response: gato, cross attention helps by

Looking into input words.
↝ giving more attention to "cat"
↝ giving less attention to "the" & "black"
(helps maintain accuracy).

* Multi heads look at the input-output relationship in d/f ways e.g 1 head might focus on subjects.
1 head might focus on adjectives.
1 head migh focus on verbs.