GenAI with LLM's

GenAI is subset of ML and the ML models that underpin genAI have learned these abilities by finding statistical patterns in massive datasets of contents that was originally generated by humans.

Foundation models (also called base models)

GPT, BERT, LLAMa, PaLM, BLOOM, FLAN-T5

- The text we pass to LLM is known as **Prompt**
- The space/memory available to prompt is called **context window**
- Output of the model is called completion
- And the act of using the model to generate text is known as inference.

Transformer learns relevance & context of all words in sentence, not just its neighbour words but every word in a sentence. Then apply attention weights to those relationships so the model learns the relevance of each word to each other word no matter where they are input

Prompt Engineering:- (In-Context Learning)

- One-shot inference: prompt with no example or label. Large LLM models are good at it.

- One shot inference:- prompt with one example and label

- Few shot inference:- prompt with multi example & label

Generative Configuration:-

- Max new tokens: it is used to control the length of the generated text.

- Greedy Sampling: the model selects the word with the highest proba for the next token at each step.

- Random Sampling: Selects word randomly

- Top-K Sampling: selects an output from the top-k results after applying random weighted strategy w.r.t probs

- Top-p Sampling: , , , : using the random-weighted strategy with the top ranked consecutive results with probs & with a cummulative proba. <=p

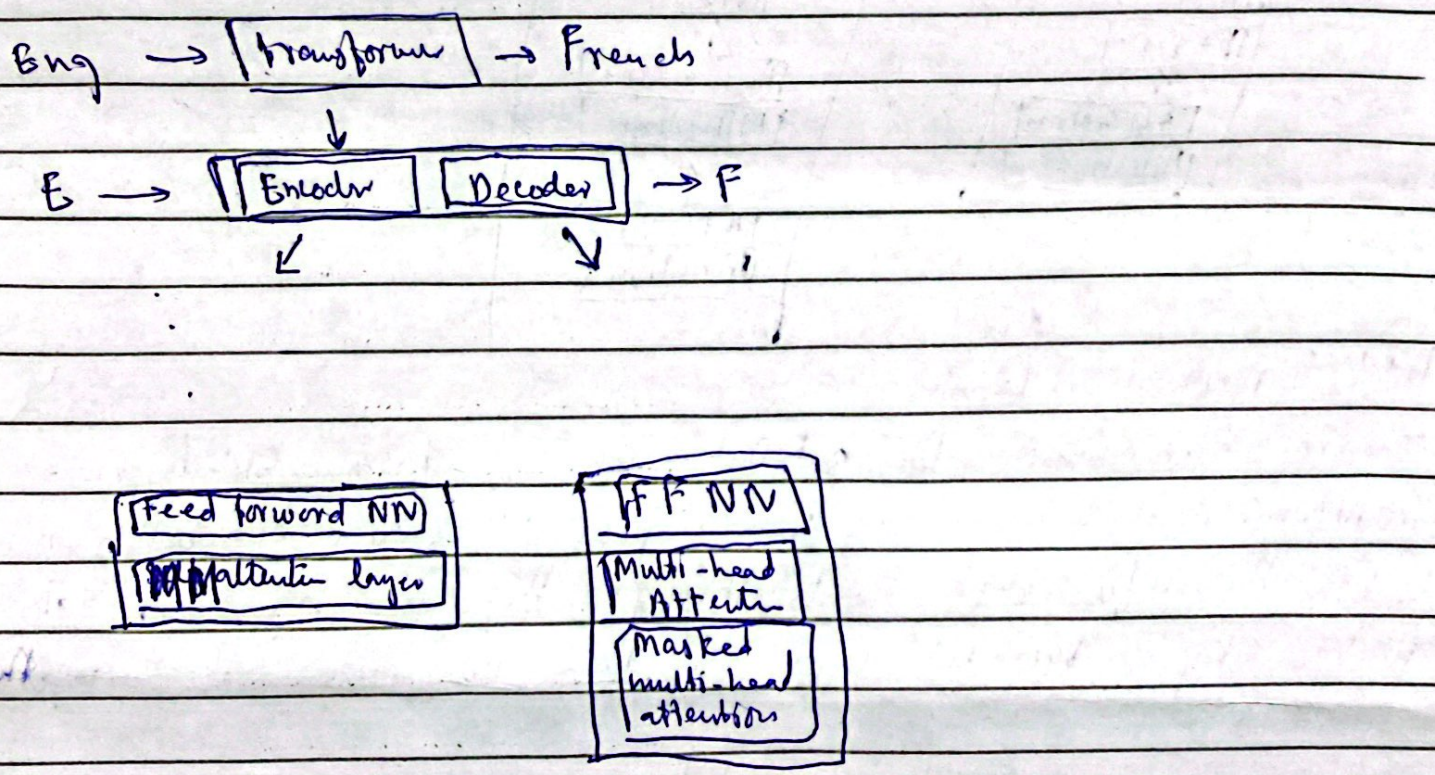
- Temp ↑ the ↑ the randomness & vice versa.

Attention: the ability of model to pay attention to important part of a sentence or image.

Transformers:-

① Input Embedding: each word is maped to a vector 512 length vector is used. But it is hyper parameter can be cha

② Positional Encoding:

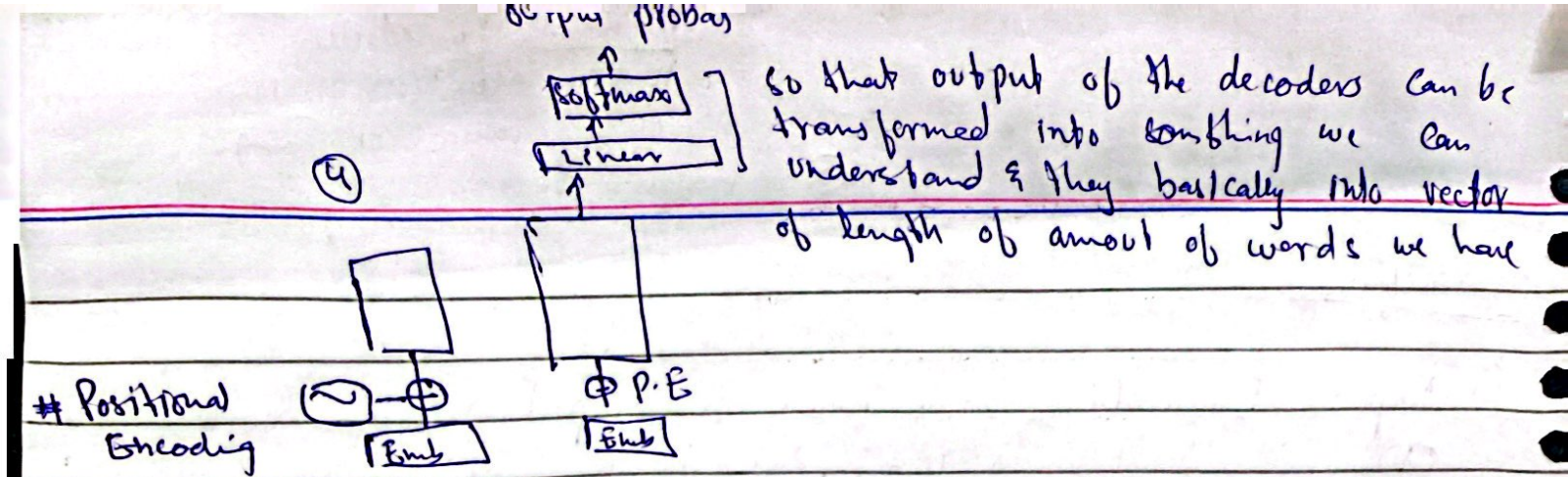


M-head Attention:-

- All the words of sentences are feed together
- All the words of the s are compared to other words in sent.

FFNN:-

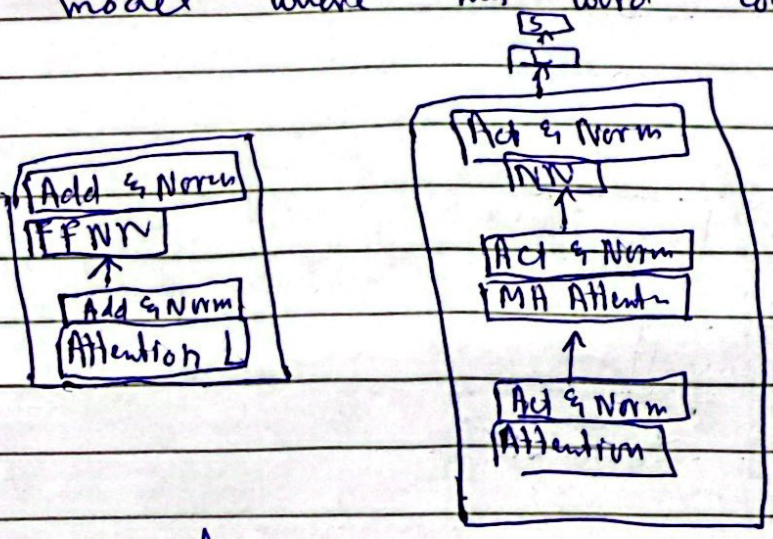
- Then they are passed through NN separately so they do not have info exchange
- NN are all different in each encoder



So that output of the decoders can be transformed into something we can understand & they basically into vector of length of amout of words we have

Positional Encoding

injecting some info about each word which tells the model where this word comes in a sentence.



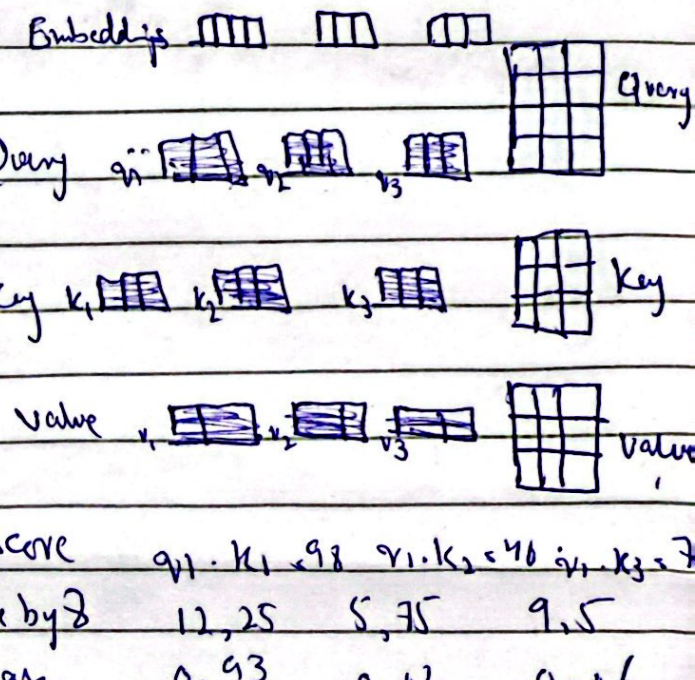
Add & Normalize layers:-

- To normalize the out put coming from sub layers. Layer Normalization is used. an improved version of batch normalization
- Add the info that went through the sub layer & also just skip the sub layer & then normalize them together

- Multi H Attention & Masked MHA are the same. but only difference is in a normal MHA L all the words are compared with all the other words but in MMHAL the the words is compared with only the words coming before it

In beginning we have embeddings of words then In attention layer these s ~~vector~~ vectors are multiplied with some matrices. These are query, key, value matrices These are values initialized randomly & are trained during training process to be learned. kind of like weights & biases. As a result we get query, key & value for each word. now we will keep these vectors to keep going with the calculations

- we will calculate score of each word against all the other words in the sentence.
- we take the dot product of query vector of each word against the key vector of all words



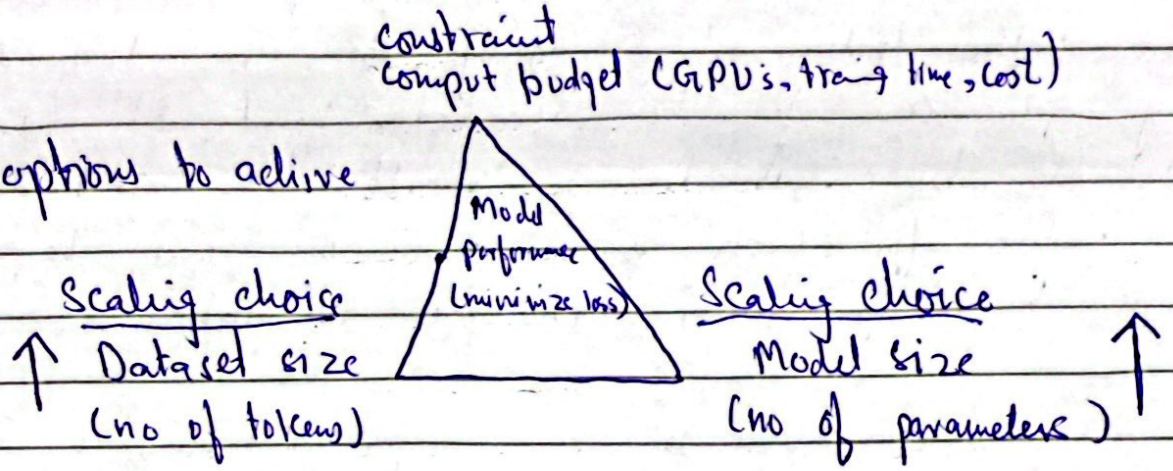
Score	$q_1 \cdot k_1 = 98$	$q_1 \cdot k_2 = 46$	$q_1 \cdot k_3 = 7$
Divide by 8	12,25	5,75	9,5
Softmax	0,93	0,07	0,06

weighted w_1 w_2 w_3

⑥

Scaling choices for Pre-training:

GOAL: maximize model performance



- Two options to achive

Types of Transformer :-

① Encoder Only: aka autoencoding models, pre-trained using Masked language model. Tokens of the input sequence are randomly masked and the training objective is to predict the mask tokens in order to reconstruct the original sentence.

- Good use cases:
  - Sentiment analysis • Named entity recognition • word classificat
- Examples models :- BERT - ROBERTA

② Decoder Only: aka Autoregressive models, pretrained on Causal language model. it predicts token based on previous sequence of tokens.

Good use cases: Text generation

Exp: - GPT . BLOOM

③ Sequence-to-sequence model : Encoder-Decoder. Masks random sequence of input tokens, then the tokens are replaced with a unique sentinal token. The Decoder reconstructs the mask token sequences

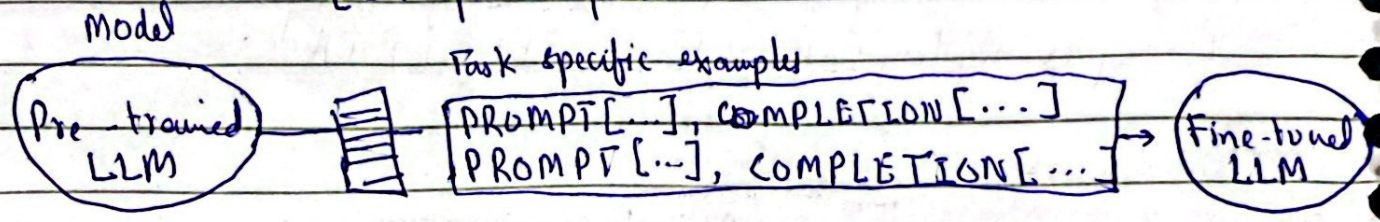
Good use Cases: Translation, text summarization, Q/A Answering

Examply: T5, BART

Fine-Tuning

- Using prompts to Fine-tune LLM's with instruction
  - if you want to fine tune your model to improve its summerization ability you'd built up a dataset of example with in which the instructions be like:

Summerize the following text:
[Example text]
[Example completion]



Catastrophic Forgetting:- when a model is fully fine-tuned to a problem, then it forget its previous trained memory. Then it does not work on other problems.

How to avoid:

- Fine tune multiple tasks at the same time.
- Consider Parameter Efficient Fine-tuning (PEFT)

LLM Evaluation - Metrics:-

ROUGE

- Used for text summerization
- Compares a summary to one or more reference summaries

BLEU Score

- Used for text translation
- Compares to human-generated translations.

ROUGE-1

- Reference (Human)
  It is Cold outside

- Generated Output
  It is very Cold outside

$$R\text{-}1\ \text{Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$R\text{-}1\ \text{Precision} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$R\text{-}1\ F1 = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

R-2: uses bigram matches instead of individual words

$$R\text{-}2\text{-}R = \frac{2}{3} = 0.67$$

$$R\text{-}2\text{-}P = \frac{2}{4} = 0.5$$

$$R\text{-}2\ F1 = 2\,\frac{0.335}{1.17} = 0.57$$

(1) Reference (Human) :

(10)

ROUGE-L : uses LCS (Longest common subsequence).

R-L Recall = $\frac{LCS(Gen, Ref)}{\text{unigram in reference}} = \frac{2}{4} = 0.5$

R-L Precision = $\frac{LCS(Gen, Ref)}{\text{unigrams in output}} = \frac{2}{5} = 0.4$

R-L F1 = $2 \frac{\text{Precision} \times \text{recall}}{\text{Precision} + \text{recall}} = 2 \frac{0.2}{0.9} = 0.44$

BLEU :

BLEU metric = Avg (precision across range of n-gram sizes

Parameter efficient fine tuning (PEFT) :

- Only updates small subset of parameters
- Most of the model weights are freezed & focus on fine tuning a subset of existing model parameters e.g particular layers or components.

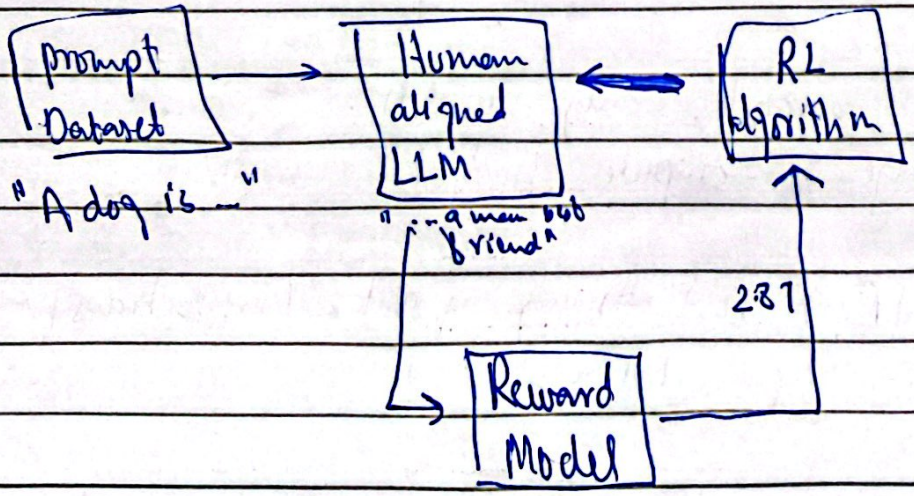
QLoRA = LoRA + quantization

(11)

PEFT technique 2: Soft prompts

- You add additional trainable tokens to your prompt &
Unlike hard prompt which are direct & specific, soft prompt are more subtle & indirect.

Fine-tuning with reinforcement learning (RLHF)



* So how fast do you need your model to generate completion?
* What compute budget you have available?
* Are you willing to trade off model performance for improved inference speed or lower storage?

* Do you intend for your model to interact with external data or other applications?
  if so how will you connect to those resources?
* How your model will be consumed

LLM optimization techniques:-

① Distillation: Larger teacher model train smaller student model. The student model statistically mimic the behaviour of the teacher model. either in final prediction layer or in hidden layers as well.

- Effective for encoder only models.
- To lower storage & compute budget.

② Quantization (PTQ): Post training quantization: transform weights to lower precision representation

③ Pruning: Remove model weights with values close or equal to zero
In theory reduces model size & improves performance. In practice only small % in LLMs are zero weights