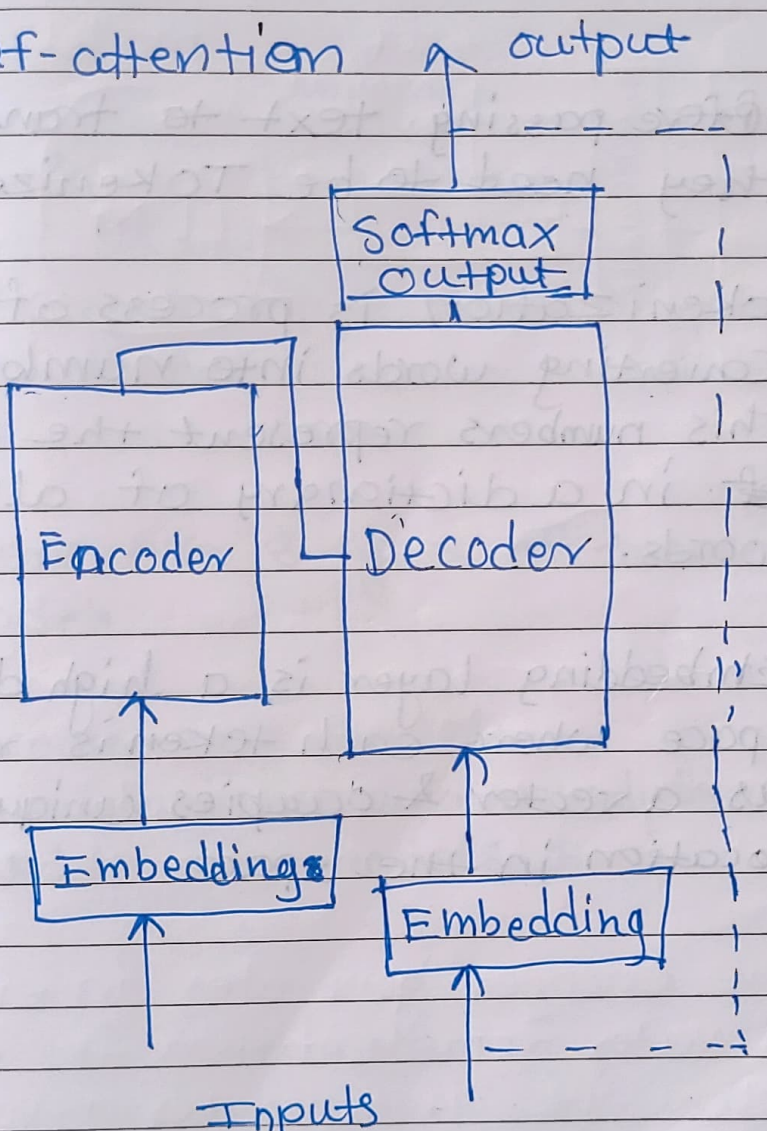
Generative AI with LLMs

* Transformers

- Self-attention



The transformer architecture is split into two distint parts,

i) Encoder
ii) Decoder

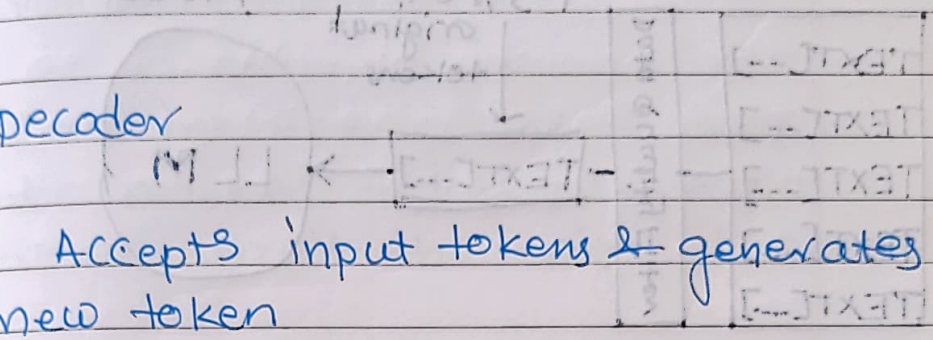
- Before passing text to transformers they need to be Tokenized.

- Tokenization is process of converting words into numbers. This numbers represent the position of in a dictionary of all possible words.

- Embedding layer is a high dimensional space where each token is represented as a vector & occupies unique location in the space.

Encoder

Encodes inputs ("prompts") with contextual understanding & produces one vector per input token.



Decoder

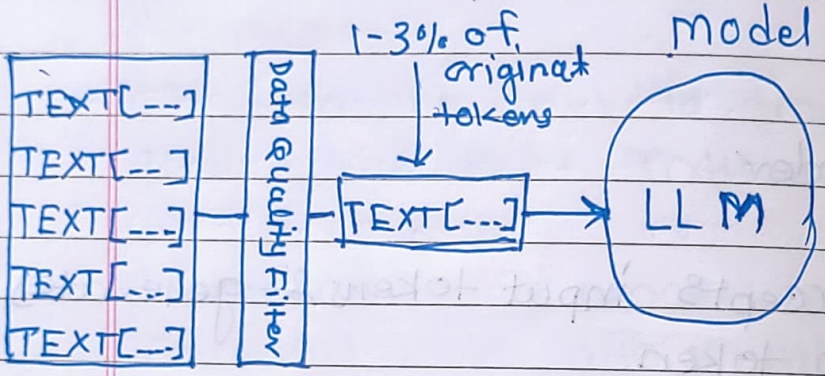
Accepts input tokens & generates new token

There are 3 types of transformer models

- Encoder only models
- Encoder - Decoder models
- Decoder only models.

* Pre training LLMs

LLM pre training at a high level



GB-TB-PB of unstructured data

- During pretraining the model weights get updated to minimize the loss.
- The encoder generates an embedding or vector representation for each tokens i.e. it generates a vocabulary
- often only 1-3% of tokens are used for training

- Encoder only models or Autoencoders

- Trained using masked language modelling

- Tokens in input sequence are randomly masked *

- Training objective is to predict the masked token in order to reconstruct the original sentence

- Also called as denoising objective.

- Autoencoding models spilled b bi-directional context.

- This means that the model has understanding of full context of token & not just the words that come before.

- Auto encoders models are good for sentiment Analysis, Named entity recognition, word classification

- ex. BERT, ROBERTA

- Decoder only models or Autoregressive models.

- This models are pretrained using causal language modelling (CLM)

- The training objective is to predict the next token based on the previous sequence of tokens.

- This is sometimes called as full language modelling.

- Decoder based autoregressive models masks the ~~to~~ input sequence & can only see the input tokens leading up to the token in question.

- The model has no knowledge of the end of sentence.

- The model then iterates over the input sequence one by one to predict the following token.

- unidirectional context.

- Autoregressive models are good for Text generation, other emergent behavior (depends on model size)

- ex. GPT, BLOOM

• Sequence to sequence models

- Uses both encoders as well as decoders

- The exact details of pretraing objective varies from model to model.

- A popular sequence to sequence model T5.

- pre trains the encoder using span corruption.

- span corruption masks random sequences of input tokens.

- Those mass sequences are then replaced with a unique sentinel token.

- Sentinel tokens are special token added to the vocabulary, but do not correspond to any actual word from input text.

- The decoder is then tasked with reconstructing the masked token sequences auto-regressively.

- The output is the sentinel token followed by predicted tokens.

- You can use sequence-to-sequence models for Translation, Text summarization, Question-Answering

- ex. T5, BART

classmate
Date
Page

* Computational Challanges.

CUDA → Compute unified Device Architecture.

- CUDA is collection of libraries & tools ~~desig~~ developed fer NVIDIA GPUS

- Approximate GPU RAM nedded to store 1B parameters

- 1 parameter = 4 bytes (32-bit float).
- A 32 bit float takes up 4 bytes of memory.
- So to store 1B parameters you'll need

4×10^9 bytes = 4GB @ 32-bit full precision.

- Additional GPU RAM needed to train 1B parameters.

model parameters (weights)	4 bytes per parameter.
Adam optimizer (2 states)	+ 8 bytes per parameter.
Gradients	+ 4 bytes per parameter.
Activations & temporary variables needed by funct.	+ 8 bytes per parameter (high-end estimate)

Total = 4 bytes per parameter
+ 20 bytes per parameter

- Approximately you'll require six times the memory needed to train the model.

for ex.

If you require 4GB of memory to store the model you'll approximat require 24GB of memory to train the model

- Techniques to reduce memory size.

- Quantization:

Reduce the precision point from 32-bit floating point to 16-bit floating point or 8-bit Integer.

FP32 → 32 bit full position

FP16 or BFLOAT16 → 16-bit half precision.

INT8 → 8 bit Integers.

Range of numbers that can be represented by FP32 goes from

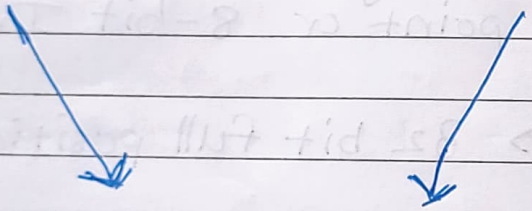
$-3e^{38}$ to $+3e^{38}$

for FP16

-65,504 to 65,504

for ex. represing pi ~~from~~ FP32 to FP16.

3.1415920257568359375 → FP32 (4 bytes)



3.140625 → FP16 (2 bytes of memory)

- A popular alternative to FP16 is BFLOAT16 or BF16.

* Efficient multi GPU training statergis

- Distributed Data parallel (DDP)
- Process batches of training data in parallel across multiple GPUs

- Fully sharded Data parallel (FSDP)
- In this technique you not only split your data you split your model (i.e. weights, optimizer layers etc) across multiple GPUs.
- Used when model is too big to train on single GPU

* Scaling laws & compute-optimal models

- Goal during pretraining is to maximize the model performance of its learning objective which is minimizing loss when predicting tokens.

- two options to achieve better performance are increasing the size of dataset you train your model on & increasing number of parameters in you model.

- Compute budget for training LLMs.

1 petaflop/s-day =

floating point operations performed at rate of 1 peteflop per second for 1 day

Note: 1 petaFLOP/s = one quadrillion floating point operations per second

classmate
Date
Page

- Relationship between compute budget, Dataset size & model size.

{ we are assumming other two parametes are held fixed when changing one. }

1)

- Dataset size & model size are held fixed so we get more model performance as compute budget increases

- But in practical cases compute budget is ~~often fixed~~..

2)

- compute budget & model size are held fixed so we get as the dataset size increases, performance of model increases.

3)

- Compute budget & dataset size is held fixed so we get as the model size increases the performance of model increases.

Note!- Both research & industry communities have published lot of papers on this exact topic trying to find ideal balance between these three quantities.

- chinchilla paper. Jordan et al 2012.

- Compute optimal models.

- Very large models may ~~have~~ be over-parameterised & under trained.

- Smaller models trained on more data could perform as well as large models.

- Compute optimal training datasize. is ~20x number of parameters.

for ex. # of parameter → 70B
therefore Compute optimal
# of tokens → 70B x 20
= ~1.4T

- LLama 65B is close to ~~no~~what Chinchilla paper reccommended

- while models like GPT-3, ~~OPT-13~~ OPT-175B & BLOOM are undertrained

* pretraining ~~for~~ domain adaptaion.

- ~~ther~~ It is generally ~~better~~ to use a pretrained model when developing a ~~software~~.

- But there are situations where pretraining ~~for~~ a model from scratch can be beneficial.

- suppose you are developing an app to help lawyers summarize legal documents.

- In this case legal documents use words which are not commonly use. This document uses very specific legal terms.

- In such case pretraing a model from scratch can be beneficial.

for ex.

BloombergGPT is a model which is domain adapted for finance.

51% → financial data

49% → public data.