LLM => BERT, GPT, Llama, Bloom, FLAN-T5, PaLM

previous generation LLM used RNN

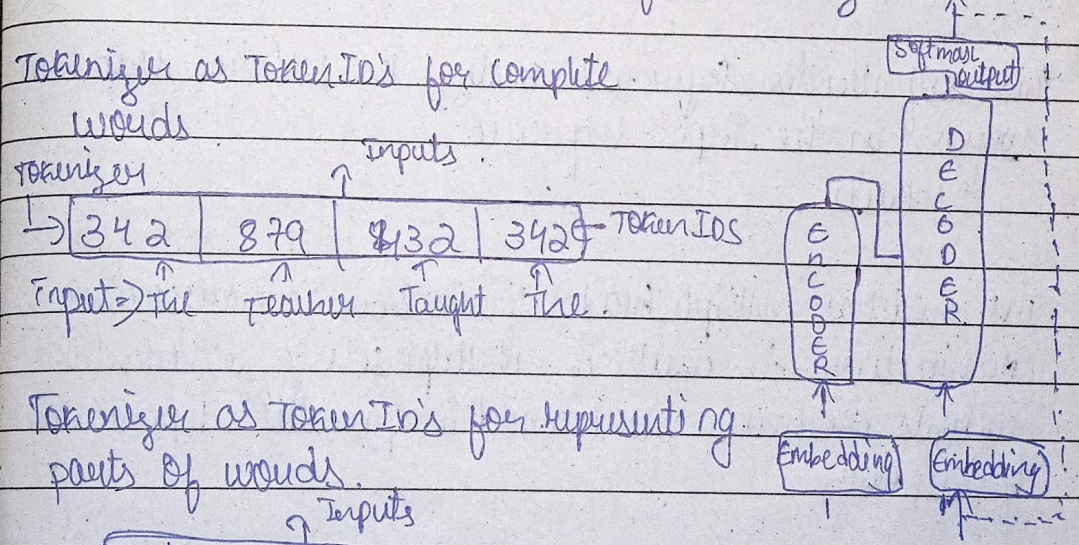
Self-attention the ability to learn a tension in this way across the whole input significantly.

Attention is all you need by google and the university of Toronto :-) gave birth to the Transformer architecture

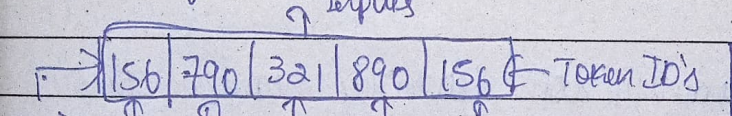
- scale efficiently, parallel process + Attention to input

"machine learning models are just big statistical calculators and they work with numbers not with words"

To work with words before passing texts into the model to process, you must first tokenize the words. Simply put the words into numbers, with each number representing a position in a dictionary of all possible words that the model to process you must first tokenize the words



Tokenizer as Token Ids for representing parts of words.



What Important is that once you have selected a Tokenizer to train the model you must use the same Tokenizer when you generate text.

Now the words or text converted to text, you can pass it to the embedding layer. This layer is a trainable vector embedding space, a high dimensional space where each token is represented as a vector and occupies a unique location within that space.

Like the words fed to the embedding layer can be plot into 3-d space and see the relationship b/w those words we can calculate the distance b/w them using angle b/w them. "This gives the ability to mathematically understand language".

adding token vectors into base of encoder & decoder will add the positional encoding of the words.
→ positional encoding - "you preserve the information about the word order and don't lose the relevance of the position of the word in the sentence." → The model process the input tokens in parallel.

summed the input tokens and positional encoding, and passing resulting vectors to the self attention layer

this self attention layer will analyze the relationship b/w words in the input sequence.
↳ Tokens

The model as multiple head self attention in transformer architecture → results in multiple sets of self attention weights or heads are learned in parallel independently of each other.

This learn independently some learn the meaning, other rhyming we can not specify to the layer what to learn they are in general 12-100 layers in transformer.

the results of weights of self attention is processed to a fully connected feed forward network → "The output of this layer is a vector of logits proportional to the probability score for each and every token in the tokenizer dictionary"

we can pass these logits to a final softmax layer where they are normalized to a probability score for each word. this output includes a probability for every single word in the vocabulary, so there's likely to be thousands of score here one single token will have a score higher than other that token is most likely to be predicted.

There are no of ways to select from this vector probabilities

and data that leaves from the encoder to decoder is a deep representation of the structure and meaning of the input sequence this is inserted at the middle of ~~the~~ decoder to influence the decoder's self attention mechanisms.

a start of sequence token is added to the input of the decoder. this triggers the decoder to predict the next token. it does based on the contextual understanding that is provided from the encoder.

the output goes to feedforward NN and through softmax layer and a ~~token~~ token is generated will continue this loop until all words then we will detokenize the words.

Transformer models

encoder only models
sequence to sequence (same length)
↳ classification tasks. sentiment analysis
· BERT.

encoder decoder
Translation different length
↳ general text generation tasks
BART.

only decoder
↳ GPT
Gratopts
Llama.

The text we give as input is known as prompt
The act of generating text is known as Inference.
The output text is known as the completion.

Context window = The limit of words that can be feed in the model.

provide examples inside the context window is a way of asking models to generate in a certain way. where it is called In context learning.
Zero shot inference one shot inference few shot inference

Configuration parameters.

Generative configuration -> greedy -> The word/token with the highest probability is selected.

Random sampling -> chose a random word/token randomly by we reduce the likelihood that words will be repeated.

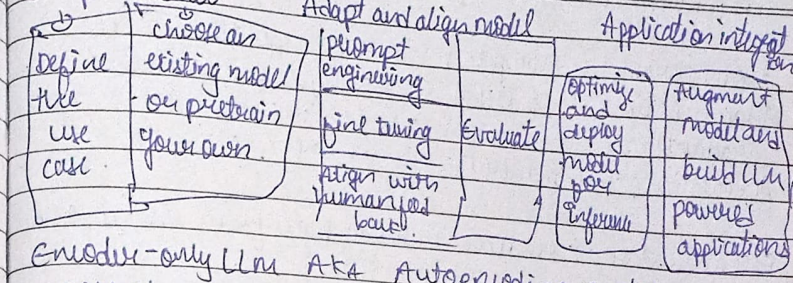
top k - sampling -> select an output from the top k results applying random weighted strategy using the probabilities.

Suppose k=10 choose only from those top 10 tokens

top p - sampling: select an output using the random weighted strategy with the top-ranked consecutive results by probability and with a cumulative probability <= p.

Temperature -> ↑ randomness higher
↓ Temperature -> ↓ randomness lower

Generative AI project lifecycle



Encoder-only LLm AKA Autoencoding models
Masked language modeling (MLM)
Objective: Reconstruct text ("denoising")
Good use cases: -> bidirectional
* Sentiment analysis, named entity recognition, word classification
models: BERT, ROBERTA.

Decoder only AKA Autoregressive models
causal language modeling (CLM)
Objective: predict next token..
Good use cases: -> unidirectional.
* Text generation * other emergent behavior.
models: GPT BLOOM.

Sequence to sequence models. AKA Encoder-decoder LLM.
span corruption. (sentinel token)
Objective: Reconstruct span.
Good use cases:
* Translation, Text summarization, question answering
Example models: T5 BART

"CUDA -> Computer Unified Device Architecture"

computational challenges of training LLMs.

1 parameter = 4 bytes (32-bit float)
~~10 parameters = 0x10 bytes (4GB)~~ → 6 times

Model parameter (weights) → 4 bytes per parameter.
Adam optimizer (2 states) → +8 bytes per parameter.
Gradients → +4 " " "
Activation & temp memory → +8 " " "

Quantization: reduce memory bytes from 32-bit to 16-bit or 8-bit integer

FP32 → FP16 | BFloat16 | INT8

FP32 [0] [00000000] 23 bits fraction.
sign 1 bit; 8 bits exponent

Mantissa/Significand = precision

~~FP32~~	Bits	Exponent	fraction	memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT	8	+/-	7	1 bytes.

To improve performance we can increase the size of the dataset and/or increasing the no of parameters in the model

constraint = compute budget (GPUs, training time, cost)

1 petaflop/s-day = 1000,000,000,000,000 (one quadrillion) floating point operations per second

Chinchilla paper →

Week-2.

Limitations of in context learning. W1-2

* In-context learning may not work for smaller models/LLMs
* Examples take up space in the context window.

pre training where you train the LLM using vast amounts of unstructured textual data via self-supervised learning.

Fine tuning is a supervised learning process where you use a dataset of labeled examples to update the weights of the LLM

Instruction fine tuning trains the model using examples that demonstrate how it should respond to a specific instruction.

fine tuning is meant as instruction fine tuning.

Sample prompt instruction templates → prompt
* classification / sentiment analysis } The result contains now contains both an instruction and the example from the dataset
* Text generation
* Text summarization.

you can pass the prompt from this above training dataset and LLM's generate a completion now we can compare it with the label (training label) so the compared result distribution of the completion & label will give the cross entropy loss. By this loss we can update our model weights in a standard back propagation. we will do this for many batches of prompt completion pairs & several epochs, update the weights so the model's performance on the task improves.

"The output of an LLM is a probability distribution across tokens".

Catastrophic forgetting happens because the full fine tuning process modifies the weights of the original LLM. while this leads to great performance on the single fine tuning task it can degrade performance on other tasks

How to avoid catastrophic forgetting.

Multitask fine tuning is a extension of single task fine tuning. Training dataset is composed of examples inputs & outputs for multiple tasks.

Multitask instruction fine tuning -> FLAN family
FLAN -> "fine tuned language net".

Model evaluation metrics.
Accuracy = $\frac{\text{correct predictions}}{\text{Total Predictions}}$

ROUGE and BLEU, 2 widely used evaluation metrics for different tasks

ROUGE or recall -> compares LLM summaries with human summaries

BLEU -> Bilingual evaluation understudy is a an algorithm designed to evaluate the quality of machine-translated text again by comparing it to human-generated translations.

In the anatomy of language, "a unigram" is a single word. A "bigram" is two words and "n-gram" is a group of n-words.

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

ROUGE-1 Precision = $\frac{\text{unigram matches}}{\text{unigram is output}} = \frac{4}{5} = 0.8$

ROUGE-1 F1 = $\frac{2 \text{ precision} * \text{recall}}{\text{precision} + \text{recall}} = \frac{2.08}{1.8} = 0.89$

F1 score is the harmonic mean of Recall & precision.

ROUGE-2 => bigram and soon for better we use the LCS: longest common subsequence.

BLEU score quantifies the quality of translation by checking how many n-grams in the machine generated translation match those in the reference translation.

BLEU & ROUGE are both cost effective to calculate we should not use them alone to report the final evaluation of a large language model.

Benchmarks = GLUE, SuperGLUE, Helm.

GLUE or General language understanding evaluation in 2018 2019 -> superGLUE

Massive multitask language understanding or MMLU.

HELM -> Holistic Evaluation of language Models
multimetric approach measuring 7 metrics across 16 core scenarios.
metrics.
1) Accuracy 2) calibration 3) Robustness 4) fairness 5) Bias 6) Toxicity 7) Efficiency.

Full fine tuning of large LLM -> is challenge so we have parameter efficient fine tuning (PEFT).
only update a small subset of parameters.
Freeze most of the model weights & focus on fine tuning a subset of existing model parameters.
or
We can add new parameters or layers to the existing LLMs. and finetune the new components.

full fine tuning creates full copy of original LLM per task & each of these is same size as the original model.

PEFT methods

Selective - select subset of initial LLM parameters to fine tune

Reparameterization - model weights using a low rank representation (LORA)

Additive : Add trainable layers or parameters to model (Adapters) (soft prompts prompt tuning)

LORA : Low Rank Adaption of LLMS

freeze most of the original LLM weights

Input 2 rank decomposition matrices.

Train the weights of the smaller matrices

$B \times A = B \times A$.

Add to original weights. $W + BXA$

In principle, you can use LORA to train for many tasks switch out the weights when you need to use them and avoid having to store multiple full-size versions of the LLM.

Soft prompt just like prompt Engineering.

The set of trainable tokens is called a soft prompt, they don't have any discrete fixed words

we can think of them as virtual tokens that can take on any value within the continuous multidimensional embedding space.

we can train different set of soft prompts for each task and then easily swap them out at inference time. just like LORA.

W3

RLHF → Reinforcement learning from human feedback

Model behaving badly

* Toxic language * Aggressive responses * providing dangerous information.

Helpful? Honest? Harmless?

Additional fine tuning with human feedback helps to better align models with human preferences and to increase these above 3 things

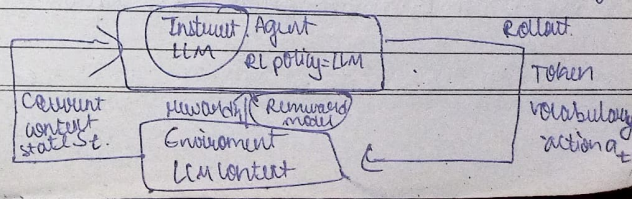
Instruct fine-tuned LLM → RLHF → Human aligned LLM.

RL - learning process is iterative & involves trail & error.

The Reward for the RLHF are given assigned based on how closely the completions aligns with human preferences.

The feedback is represented in a scalar value zero or one LLM weights are then updated iteratively to maximize the reward obtained from the human classifier.

The sequence of actions and states is called a rollout, instead of playout



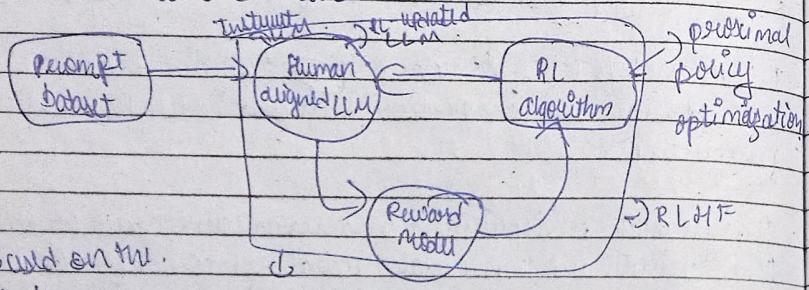
collect human feedback.
- define your model alignment criterion
- for the prompt-response sets that you just generated, obtain human feedback through labeler workflow.

The LLM will generate different completions based on the order of helpfulness from the most to least rank them using labelers.

train the model with all the data that is labelled by human labellers, this data then can be classified without humans.
the feed back thump us & thumbsdown is effective way to gather ranking.

Ranked feedback gives you more prom completion data to train your reward model.

reward model is a binary classifier to provide the positive & negative classes.
reward value.

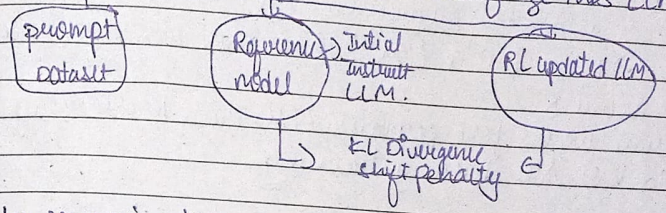


based on the feedbacks reward model gives reward values based on it RL algo will update our fine tune weights and repeat till no of epochs

PPO → Proximal policy optimizations updates the weights of the model(RL) which is very near to the previous version of LLM.
"keeping the changes within this small region results in a more stable learning".

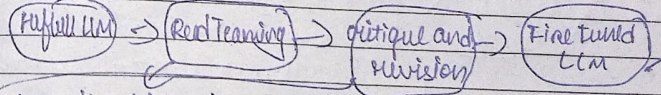
Reward hacking where agent learns to cheat the system by favoring actions that maximize the reward received even if those actions don't align well with the objective.

to avoid the reward hacking we can use the reference model of a initial instruct LLM and freeze this LLM.



by comparing the completions of this two models we get the difference value that is called "kullback-leibler divergence" KL divergence.

"constitutional AI is one approach of scale supervision." (self supervision).



prove the LLM to generate harmful content/completion review or act as a critique to it own responses by considering the constitutional principles.

Model optimizations for deployment.

Distillation → encoder only model
- train a smaller student model from a large teacher model

post training quantization (PTQ).
FP32 → FP16 | INT8 | BF16 | int4

pruning
Removing model weights with values close or equal to zero
→ full model re-training; PEFT/LORA, post training.

Quantization, Distillation and pruning all aim to reduce model size to improve model performance during inference without impacting accuracy.

Retrieval Augmented Generation.
RAG → helps build LLM that can make use of external data sources.
→ framework for providing LLMs access to data they did not see during training.

Chain of thoughts human way of thinking.

PAL → program aided language models.
→ framework that helps LLM to interact with other languages (Python interpreter)

RAG → Query encoder → External information source

External Information
* Documents, * wikis
* Expert-systems, * Webpages
* Databases * Vector stores

Retrievers
* Lang chain to map the split of small chunks of external data to handle token limit
* Embedding vectors.

Chain of thoughts Humans take a step by step approaches to solving complex problem.

User application ⇄ Orchestration library ⇄ External data sources / External Applications

ReAct: is a prompting strategy that combines chain of thought reasoning with action planning
* first, the task is defined telling the model to answer a question using the prompt structure you just explored in detail
* next, the instructions give more detail about what is meant by thought and the specifies that the action step can only be one of three types (for this example)

Aws sagemaker Jumpstart is a model hub it allows you to quickly deploy foundation models that are available within the service & integrate them into your own applications.