GENERATIVE AI WITH LLMs

WEEK 1

What's included?

Introduction to LLMs and GenAI Project lifecycle

1. Transformers architecture
2. Generating text with transformers
3. Paper discussion: Attention is all you need
4. Prompting and prompt engineering
5. Generative configuration
6. GenAI Project lifecycle

LLM Pre-training and scaling laws

1. Pre-training large language models
2. Computational challenges of training LLMs
3. Scaling laws and optimal models
4. Pre-traing for domain adaptation
5. Discussion: Bloomberg GPT

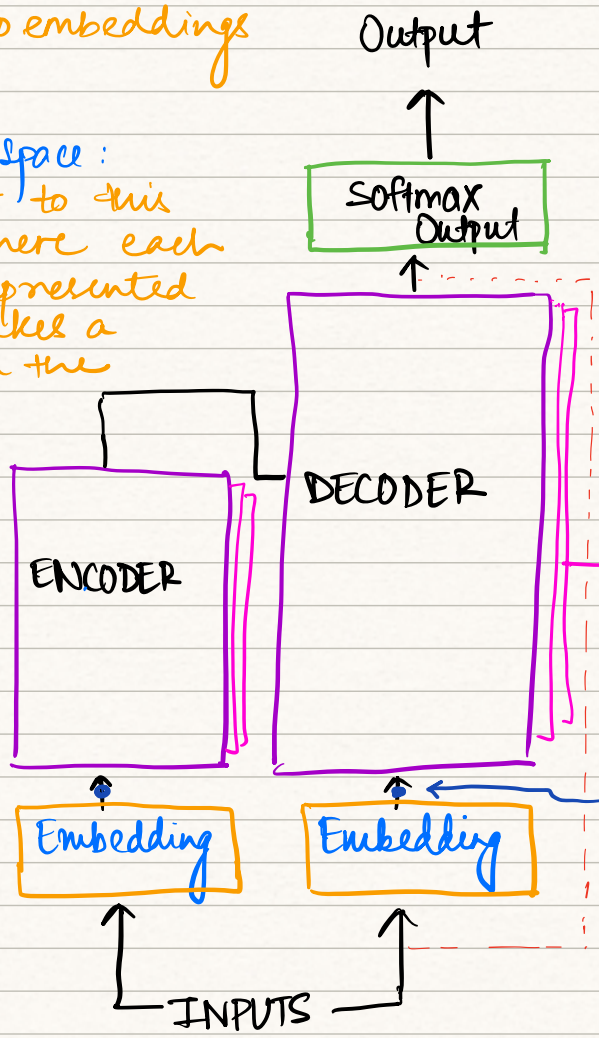
TRANSFORMER ARCHITECTURE

SELF ATTENTION: learn relevance of each word with every word in a sentence

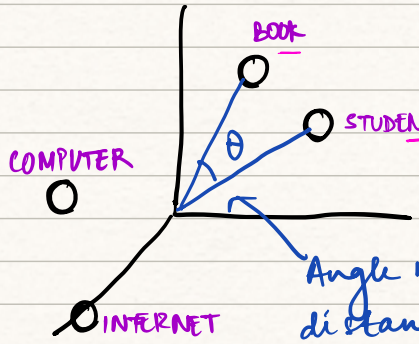
1. Tokenizer:
Convert numbers into embeddings (numbers)

2. Vector Embedding Space:
Each token is sent to this embedding layer where each token is represented as a vector and takes a unique location in the multidimensional vector space.

3. Meaning & Context:
In this space, the tokens learn meaning & context w.r.t each other.
WORD2VEC is an example.



4. WORD → TOKEN → VECTOR.



Angle measures distance between the words, based on which the model learns context & meaning.

5. Positional Encoding:
The model processes each encoding in parallel. This encoding preserves meaning as well as position of a word ∴ the word order so that relevance of a word w.r.t its position in the sentence is not lost.

6. SUM INPUT TOKEN AND POSITIONAL ENCODINGS. AND GET RESULTING VECTORS (VECTOR = VALUE (INPUT TOKEN) AND DIRECTION (POSITIONAL ENCODING).

7. Pass resulting vectors into self-attention layers:
- The model analyzes the relationship between tokens in the input sequence. All words are accessible to all other words in the sentence to better capture the meaning & context of each ∴ Attend to each other. and learn dependency between all words.

- Multiheaded self Attention:
  - Multiple sets of attention weights learn themselves in parallel.
  - Model's attention head can vary from architecture to architecture (Ranges from 12 to 100)
  - Different heads learn different aspects of a language.
    Eg. • Relationship between people entities in a sentence
    • Activities in a sentence.
    • Rhyming words.

- weights are randomly initialized

8. NOW, ALL ATTENTION WEIGHTS HAVE BEEN APPLIED TO THE INPUT DATA

9. Output Processed:
Happens through a fully connected feed forward network.
Output of this layer is a vector of logits proportional to a probability score corresponding to each token in the tokenizer dictionary.

10. Pass logits to a final softmax layer:
- They are normalized into a probability score
- Highest probability token is the predicted token.

GENERATING TEXT WITH TRANSFORMERS

Translation Task
FRENCH ⟶ ENGLISH

1. Tokenizer / Embedding layer
   1. Takes in the input sentence
   2. Tokenize the words by using the same tokenizer that was used to train the network

2. Encoder:
   1. Tokens added as input to the encoder
   2. Passed through embedding layer
   3. Fed into multi-headed attention layers
   4. Outputs of this layer, fed into a feed-forward network.
   5. Data at this point is a deep representation of structure & meaning of the input sequence.

3. Decoder:
Triggered to predict the next token based on the contextual understanding provided from the encoder.

+ START OF SEQUENCE TOKEN → What is this?

SELF-ATTENTION LAYERS → output → FEED-FORWARD NETWORK → output → SOFTMAX OUTPUT

Continues in a loop, until model predicts an end-of-sequence token.

ENCODER ONLY MODELS

BERT Model

- When input sequence length = Output sequence
- Classification tasks: Sentiment analysis (additional training)

ENCODER DECODER MODELS

- length(i/p seq) ≠ len(o/p seq)

BARD, T5 Models

- Text Generation tasks (scale)
- Seq to seq tasks - Translation

DECODER ONLY

GPT, BLOOM, JURASSIC, LLAMA

PROMPT AND PROMPT ENGINEERING.

Prompt: Input fed into network

Inference → Act of generating the text.

Completion → Generated text

Zero - Shot → No eg

One - Shot → 1 eg

few - Shot → 1+ eg.

CONFIGURATION PARAMETERS.

(1) Max New Tokens - No. of tokens to be generated

(2) Sample Top K - limit Random Sampling (Top K elements to be chosen)
P - Total probability

(3) Temperature - shape of normalized probability, controls randomness.

Random Sampling

Randomness & Temperature

COOLER TEMPERATURE (<1)

prob	word
0.001	apple
0.002	banana
0.400	cake
0.012	donut

Strongly packed probability distribution

HIGHER TEMPERATURE (>1)

prob	word
0.040	..
0.080	..
0.150	..
0.120	..

Broader, flatter probability distribution.

Less variability → more strict.

More variability → more creativity

GENERATIVE AI PROJECT LIFECYCLE

1. SCOPE → Define use-case
2. SELECT → Choose existing or pre-train your own
3. ADAPT AND ALIGN MODEL → Prompt Engg., Fine Tuning, Align w human feedback, Evaluate
4. APPLICATION INTEGRATION → Optimize, deploy model, Augment & build app.

PRETRAINING LARGE LANGUAGE MODELS.

- Model trained on vast amounts of unstructured data.
- Self-supervised learning – model internalizes the patterns and structures present in the language (during training phase)
- Model weights get updated to minimize the loss of the training objective.
- Increase quality, address bias and remove other harmful content from data
  ∴ 1-3% of all tokens used during pre-training

Encoder-only models: Aka Autoencoding models

- Pre-trained using masked language modelling
- Training objective: predict the masked tokens to reconstruct the original sentence
  Aka denoising objective
- Bidirectional representation of the input sequence; model understands full context and not just the words that come before.
- Used for:
  - Sentence classification tasks
    - → Sentiment Analysis
    - → Token-level tasks – Named Entity Recognition
      – Word classification
    - → Eg. BERT, RoBERTa

Decoder-Only Models Aka Autoregressive Models

- Pre-trained using causal language model
- Training objective: predict next token based on previous set of tokens
  Aka Full Language Modelling
- Context is unidirectional ∴ model has no knowledge about End of Sentence token.
- Eg. GPT, BLOOM – Models used for text generation
  zero-shot inference abilities
  Perform a range of tasks well

Sequence-To-Sequence Models - Both Encoder & Decoder models

- Training objective varies from model to model
- T5: Trained encoder using span corruption: masks random random seq. of input tokens.
  ↓
  These tokens replaced by Sentinel Tokens
  ↓
  Added to vocabulary, but do not correspond to any actual word from the input text
  ↓
  Decoder then asked to reconstruct the sentinel token autoregressively.
  ↓
  Output: Sentinel Token followed by predicted tokens.
- Work well for translation, summarization and Q&A

Larger the models, don't need robust in-context learning or further training.

COMPUTATIONAL CHALLENGES OF TRAINING LLMs

- Running out of memory issues

PROBLEM: 1 parameter = 4 byte (32-bit float)

1B parameters = 4×10^9 bytes = 4GB

All factors considered; optimizer + weights + gradients

Total memory required = 24GB

SOLUTION : Quantization

Reduce the memory required to store the weights of model by reducing their precision from 32-bit to 16-bit floating point numbers.

or 8-bit integers

Corresponding data types in DL :

FP32 FP16 BFloat16 8-bit integers

→ Range from -3×10^{38} to 3×10^{38}
→ Model weights, activations, other model parameters stored in this format

FP32 → lower precision space, using scaling factors
(32 floating point)

↳ Eg. Store pi : 3.141592

FP32 → 4 bytes

0 | 10000000 | 10000000000011111010111000

Sign: 1 bit | Exponent: 8 bits | Fraction: 23 bits (Mantissa)

FP16 → 2 bytes ← More optimal (memory halved)

0 | 10000 | 1000010001000

sign: 1 bit | Exponent | Fraction: 10 bits

BFLOAT16 → Hybrid between FP32 and FP16
→ Training stable
→ Supported by GPUs
→ Truncates the fraction & retains the exponent part

BFLOAT8 → −128 → +127
→ 4 bytes → 1 byte

Key Points :-

1) Reduce required memory to store & train models
2) Projects original 32-bit floating point no into lower precision spaces
3) Quantization-aware training (QAT) learns the quantization scaling factors during training

SCALING CHOICES FOR PRE-TRAINING

Scaling choices for pre-training

GOAL DURING PRE-TRAINING: MAXIMIZE MODEL PERFORMANCE

- ↑ Increase dataset size
- ↑ Use large no. of parameters
- GPU performance, Time constraint → Constraint.

Compute Budget : DataSet Size : Parameters size (model)

What is the ideal combination of these to ensure high quality training?

→ CHINCHILLA PAPER

→ Key Findings:

1) larger model may be over-parametrized and under-trained.
2) Smaller models trained on more data could perform as well as large models.

Chinchilla scaling laws for model and dataset size

Model	# of parameters	Compute-optimal* # of tokens (~20x)	Actual # tokens
Chinchilla	70B	~1.4T	1.4T
LLaMA-65B	65B	~1.3T	1.4T
GPT-3	175B	~3.5T	300B
OPT-175B	175B	~3.5T	180B
BLOOM	176B	~3.5T	350B

Compliant with chinchilla

Compute optimal training datasize is ~20x number of parameters

Might be under-trained

PRE-TRAINING FOR DOMAIN ADAPTATION

- Domain specific pretraining to consider only domain-related words
- For eg. legal, journalism, medical conditions, pharmacy, etc

BLOOMBERG GPT

→ Trained on financial and public data
→ Followed chinchilla laws
→ 230,000,000,000 petaflops

END OF WEEK 1.