DEK 1:

Language model, probabilistic model that assign prob to sequence of words. P(this/context).

Base models can be used using Finetuning/Prompting.

Forcing LLM. (seq-seq model)

- input → [SOS] i/p sentence (ID) → T. Encoder → output sentence (EOS)
  No of input token = No of output token.
- Then we calculate cross entropy loss b/w output/target and we use backpropogate to adjust.

= Inference.

prompt → forcing tokens → [SOS] → [PTM] → output token

Embedding

token → ID → Embedding + PE

→ Embedding (words represented in high dimension by vector) to capture meaning

eg for 2-Dimen

⇒ The angle b/w similar word is small and dot product is big

$$PE_2(pos, 2i) = \sin\left(\frac{pos}{1000^{\frac{2i}{d_{small}}}}\right)$$

or

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{1000^{\frac{2i}{d_{small}}}}\right)$$

# BERT - overview.

## BERT - Base

- 12 encoder layers
- 12 attention heads
- 3072 Feed forward size
- 768 d_model

## BERT - large

- 24 encoder layers
- 16 attention heads
- 4096 Feed ___.
- 1024 d_model

- linear head changes according to application
- PE (768) are absolute and learnt during train & 512 pos
- used Piece tokenizer (size ≈ 30,000 tokens)
- max len (512 tokens)

## - BERT vs GPT Layer

- NA for chat but for finetuning
- NA like other, trained on left-right context.
- NA specifically for text generation.
- not trained on next token prediction, but masked LM & sentence prediction task.

# RAG (Lemon-Juice)

very useful for Q/A
- used in Gork.
- can access latest data.

- LLM assigned probab to sequence of words
- Inference, create a prompt to do so.

## Finetuning Cons:

→ expensive
→ no of parameters might not be enough to capture all the knowledge we want to teach.
→ Finetuning is non-additive. Eg* (heavily finetuned might forget some previous knowledge)

Few-Shot (prompting) to rescue: → LLM can be taught to perform new task using prompt engineering

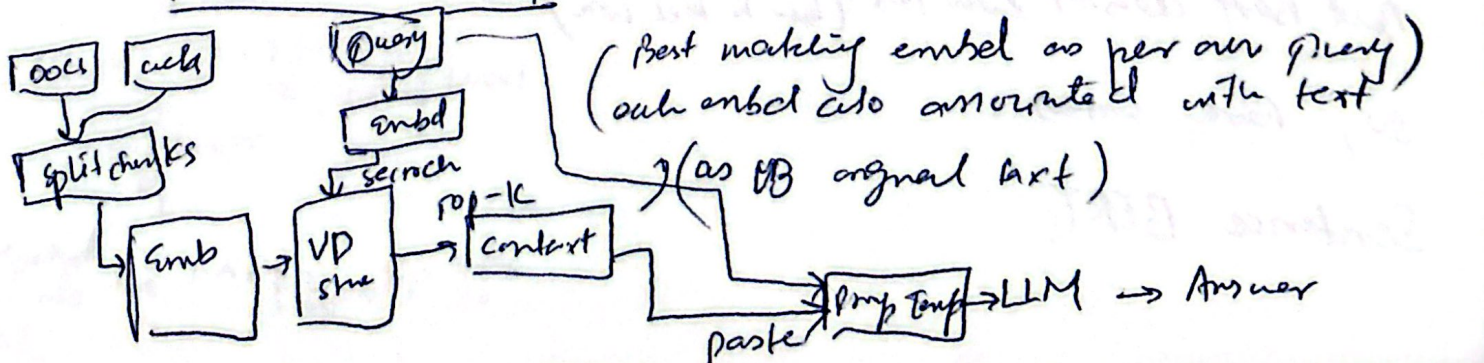
## Q/A with prompt Engineering. {instruction} {context} {Q} {A}

Answer, using access to the context on which it's not trained on

## Pros of Finetuning.

→ higher quality result as compared to prompt
→ smaller context size (input) during inference (do need context)

## Q/A with RAG:



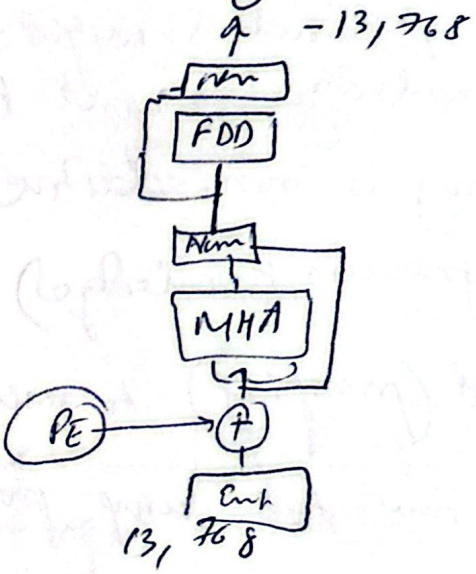
Embedding

→ Similarity by Cosine.

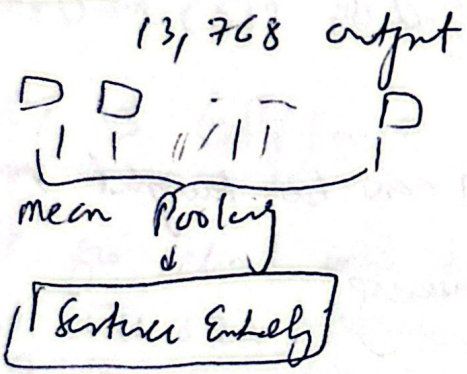
words that are synonyms tend to occur in same extend (distributional hypothesis). To capture meaning we need to have access to context.

Self-attention access 2 things word emb and position embedding and modify embedding to capture context as well

→ In backprop input embeddings are also modified.




Sentence - Embedding with Bert.




Sentence Embd comparison

By Cosine Similarity → cosine of angle btw 2-vectors.

$$Sc(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{N} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{N} A_i^2} \sqrt{\sum_{i=1}^{N} B_i^2}}$$

2 similar sentence should produce same embedding

But BERT doesn't know this (how to feel um)

So, here comes

Sentence BERT

ort-Pretraing.

## Masked language Model task (Cloze)

Randomly select words in a sentence and masked it and let model predict words based on left/right context.

left/Right context in BA

- we dont mask next value after ($\Phi \times \vec{K}$), so each token attend token to its left and right.

- pretraing procedure selects (15%) of tokens from a sentence
  - 80% by [mask]
  - 10% by zebra
  - 10% Not ag (same word)

Next-Prediction Task (NSP)

Input Emb → + Segment Emb → + PE

Represent this token is Sent A Word or B


- Input [CLS] ▢▢ ... SEP ... SEP
  Sentence (A) B (Sent)

outp [1st] [ ]

↓

Linear layer (2 features) → Next & Invat

+

Softmax

↓

Target → CE loss ↑ backprop

[CLS] token in BERT: capture-like info from all other tokens because not masked in (QKV).

BERT - Fine tuning.

(prompting technique.
one/few shot, chain of thought, ...

→ Text-classification
[CLS]

→ Q/A :

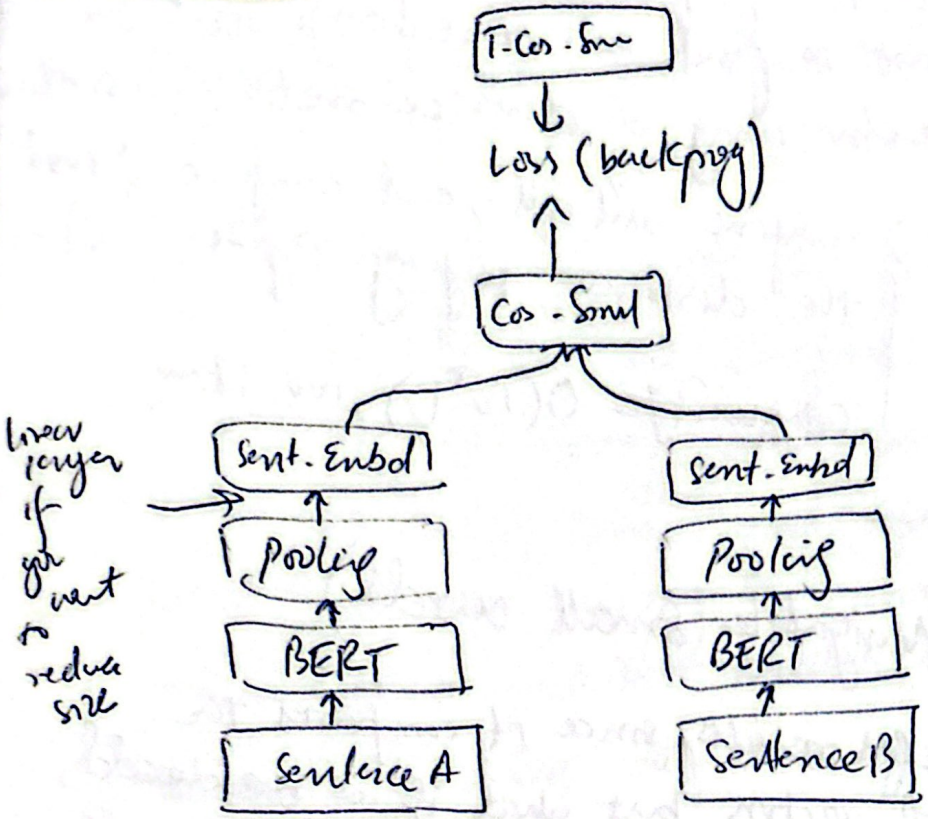
2-Issues

→ which part of input is context, which is question?
Ans: segment analysis → Q → A, C → B

→ Start & End of Answer
[CLS] → (start token or end token) → we know [this is answer]

# Sentence in BERT

## Architecture



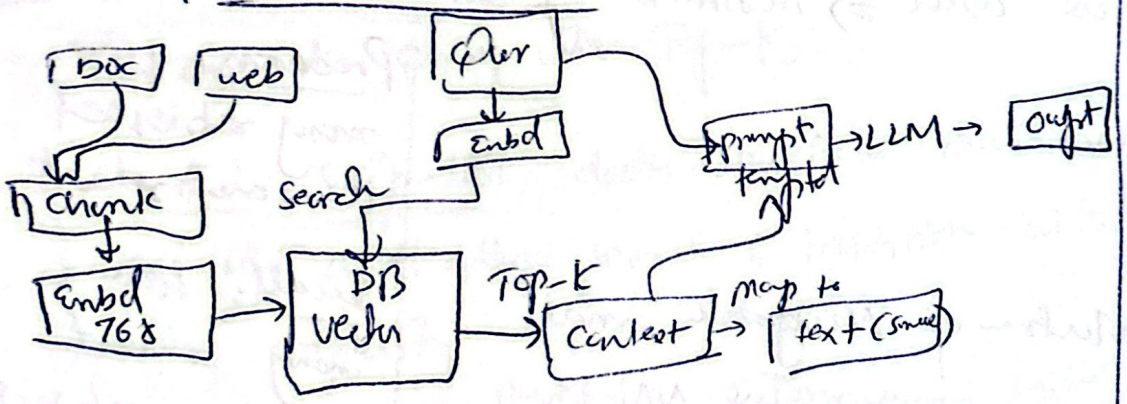

Siamese Network

→ same architecture
→ same parameters
→ same weights

Implementation

Just one network, other is branch

Q/A with RAG:




Strategies

- Fine tune
- RAG
- Fine tune + RAG

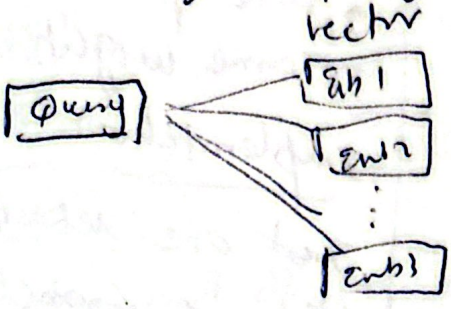
gradient platform -

Search - cosine similarity have high values - Euclidean distance < quer.

## — Vector DB:

→ Similarity algorithm → (KNN variant)

→ vector database store vector of fixed dimension such that we can query the database to find the embeddings that are closest to given query vector using a distance metric (cos similarity)




compare with all, and compare & sort the distance, keeping Top-K.

complexity = $O(N^2 D)$, too slow

## HNSW: (Heirarchical Navigable Small words)

→ always produce accurate results, since it compares the query with all stored vectors but what if we ~~see~~ decrease the no of comparison but still obtain accurate results

→ The metric we used ⇒ in similarity search is called Recall.

**Precision:** How many retrieved item are relevant

**Recall:** How many relevant items are retrieved

### Idea #1:

HNSW is an evolution of Navigable small words algorithm for approximating NNeighbors which is based on the concept of fix degree of separation

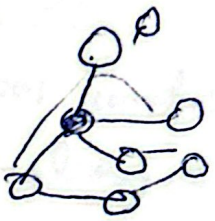
six-degree of Perception.

Navigable-Small world → it builds a graph that connects close vectors with each other but keeping no of connections small (every vector maybe connected upto 6 vectors)

→ Randomly choose a point in graph & compare similarity score with query. we compare query similarity with the friends of entry point and move which has better similarity score. and (repeat). Stop at local best (best score) among its connection (neighbours. order them best to least-best matches & keep best.

→ Repeat by doing diffent entry points

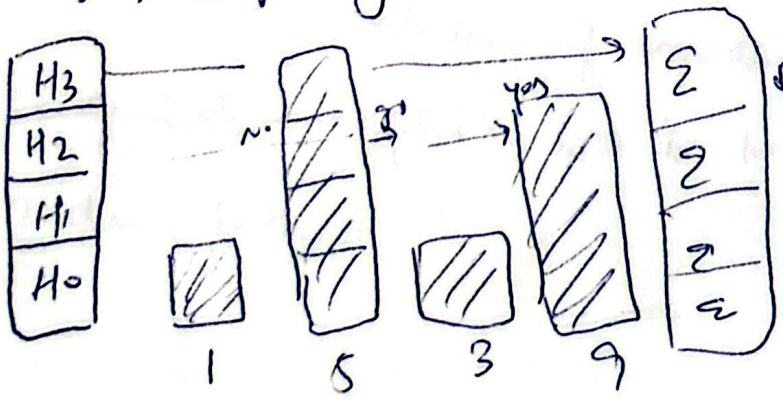
→ Keep Top-K



→ Inserting a new vector.

→ connect query with Top-K

Idea #2: Skip-list. data-structure that maintain a sorted list and allow search & insertion with an average of O(log N) complexity.




→ high layer more jumps

→ lower more work.

HNSW: Skip list + Navigable ... dense / Sparse

→ Find random entry point at top-level

→ Calculate Cosine-Similarity with it & its friends.

→ local best → go down

→ we repeat with other random starting

→ we sort & keep Top-K.