04-09-25

Large Language Model (LLM)

Book: Hands-on LLM by O'REILLY

⇒ Language Model (Generalized Way): Models that recognize, summarize, translate, predict & generate text. A language model predicts the next word in a sentence by understanding the context & based on the probability of words following each other.

⇒ Why language Model is called Large Language Model: Because they are trained on large amount of data & have many parameters.

⇒ Large Language Model:
- It is a type of AI model based on deep learning architecture, typically the transformer architecture that is trained on massive corpora/corpus/database/dataset of natural language text.
- It's primary objective is to learn the probability distribution over sequence of words or tokens, enabling it to generate coherent & contextually to relevant text.

⇒ Type of Language Model: There are two types:
i) Representation Model: These are LLMs that don't generate text but are commonly used for task specific use cases like Classification, Clustering, Regression, etc (understands the data).
ii) Generative Model: These are LLMs that generates text like GPT models. (generates the text) [O/P]

⇒ Difference between Search Engine & ChatBot:

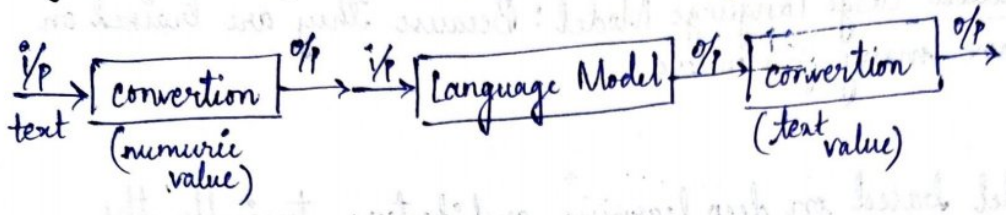
	Search Engine	ChatBot
Purpose:	It helps users to find inf^n on the vast expanse of the internet.	• It engage in conversation with user, answer, questions & often complete tasks
Output:	It returns list of relevant web-pages or links allowing users to browse & explore	• It provides a single, summarized answer & sometimes offering flow of questions
Interaction:	These provide a wide range of inf^n.	• These engage in a conversation.
Context:	These have short-term memory.	• These can maintain context across multiple interaction.
Method:	It uses algo. to crawl index & rank web pages based on keywords.	• It employs natural language processing & ML to understand user intend & provide responses.
Eg:	Google, Yahoo	• ChatGPT, Gemini, Copilot, DeepSeek

Chapter-01

⇒ **Language AI**: It refers to sub-field of AI that focuses on developing technologies capable of understanding, processing & generating human languages.

05-09-25

⇒ **Bag of Words (BoW) Representation**: [Vocabulary of Words / Term Vocabulary / Dictionary]



S_1: That is a cute dog. S_2: My cat is cute.

Tokenization ① → S_1: 'That', 'is', 'a', 'cute', 'dog'
S_2: 'My', 'cat', 'is', 'cute'

② Construct a Vocabulary

	That	is	a	cute	dog	My	cat	
S_1:	1	1	1	1	1	0	0	→ Vectors of Sentence 1
S_2:	0	1	0	1	0	1	1	→ Vectors of Sentence 2

- BoW model is a simple & widely used technique for text representation in Natural Language Processing (NLP).
- It transforms a sentence or document into a vector of word counts, ignoring grammar, word order and context. Each unique word in the corpus becomes a feature & resulting vector reflects the frequency of those words in the given text.

Limitations:

- BoW although an elegant approach but has some flaws. It considers languages to be nothing more than an almost literal BoW & ignores the semantic nature / meaning of texts & also it is represented as Sparse Matrix/Vectors.

06-09-25

⇒ **Representation of sentence as Dense Vector Embedding**:

- In Natural Language Processing, a Dense vector representation of a sentence is of fixed size numerical representation that captures the semantic meaning of the entire ~~sentence~~ sentence.
- Unlike, simple models like BoW, which only count word occurrences where Dense vectors encode relationship between words & their context where each word is converted into a vector using Word embedding model Word2Vec

→ **Word Embedding**: It represents words as numerical vector that captures the meaning & relationship between the words.

• Dense representation are foundational to have modern language models can understand & generate human language.
• A Dense Vector representation in a sentence is a compact, information reach numerical encoding that capture semantic meaning of a sentence.

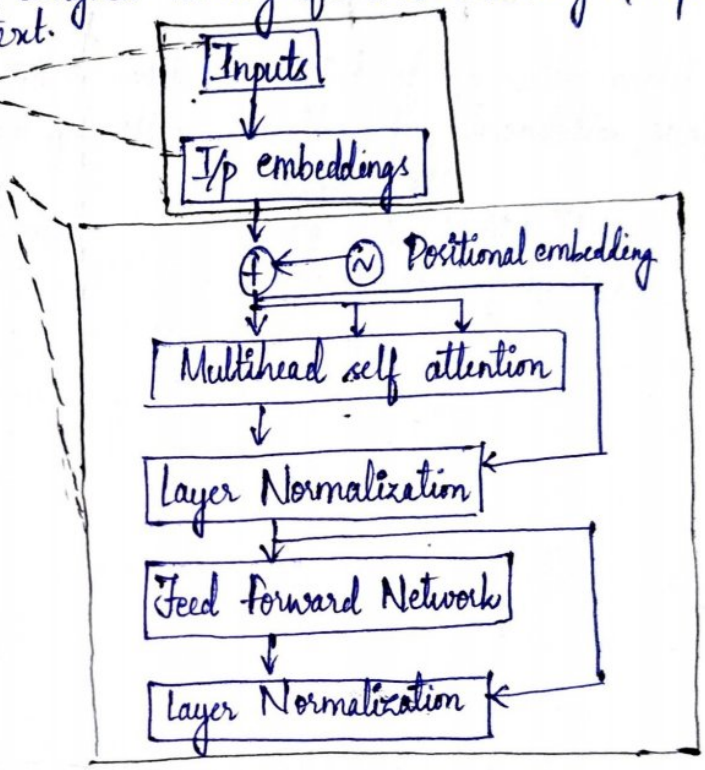
⇒ Transformer based Model
- Encoder only (BERT) - Representation type model
- Decoder only model (GPT) - Generative type model
- Encoder-Decoder model - Generative type model

⇒ Language based Model
- Representation
- Generative

⇒ Representation Model: Encoder only -

Encoder only model are type of Transformer architecture that focus on the encoder part of the original transformer design. They are designed mainly for understanding & representing input text but not generating new text.

Steps involved: Input representation → Stack of encoder layer → Output representation



08-09-25

This above diagram is based on architecture of ANN: Input layer, hidden layer & output layer.

⇒ Input Representation:
- Here the input sentence is split into tokens.
- Each token is converted into a vector by combining token embedding or word embedding or positional embedding.

⇒ Stack of encoder layer:

The vectors pass through multiple identical encoder layers & each layer has -

i) Multihead self attention: Here, each token attends to all tokens in the input to capture the relationship.

ii) Feed Forward Network: It is used to learn complex patterns & relationship in the input data.

iii) Layer Normalization: Layer normalization & residual connection helps stabilizing the training & improve gradient flow.

⇒ Output Representation:

After passing all encoder layer, the model outputs are reached vector representation for each token.

11-09-25

⇒ Representation Model: Decoder only -

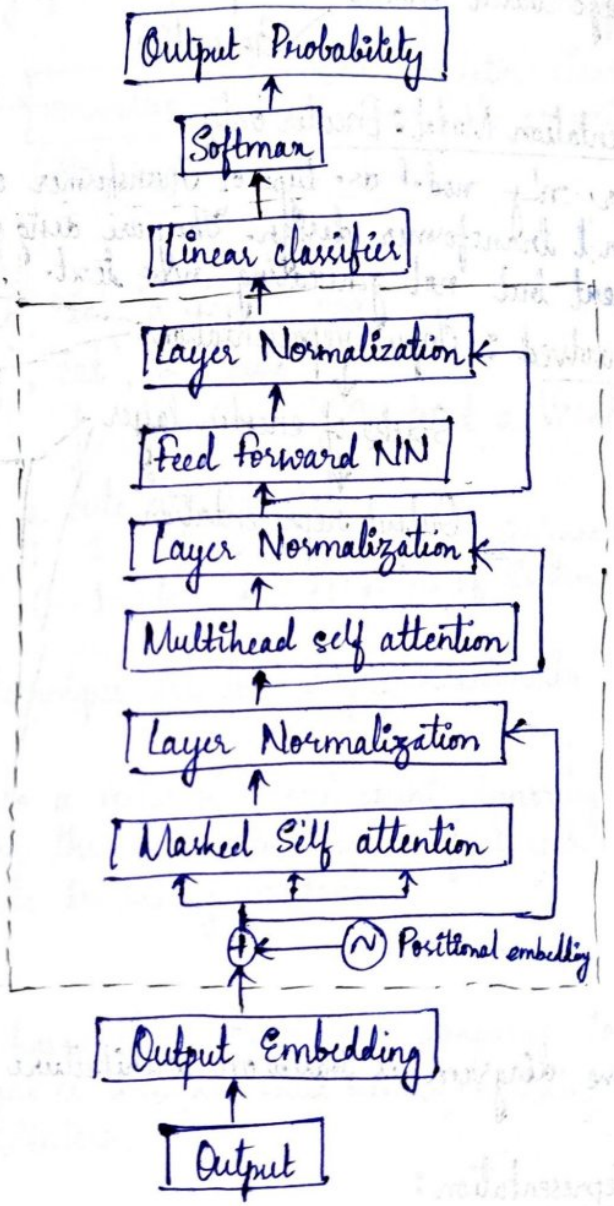
Steps involved: Input representation

↓

Stack of decoder layers

↓

Output representation



Decoder only models are transformer architecture that use only decoder part of the original transformer. They are designed mainly for generating text (1 token at a time)

13-09-25

- Masked Self attention: It is used to ensure that the model can't attend the future tokens during training.
- Feed Forward NN: It is used to transform the encoded representation back into the original input space (or) generate new data.

• Linear classifier: It computes raw scores for each class.

• Softmax: It converts the previous output into a probability distribution when a highest probability is choosen as predicted class.

Encoder only

- Models focus on input rep"
- Working Model -

I/p

↓

Encoder stack

↓

o/p representation

- Main purpose is representation learning, i.e, understanding i/p but not generating text.
- Kind of representation model.
- Used for classification & feature extraction.
- Eg: BERT, RoBERTa; DistilBERT

Decoder only

- It focus on output generation
- Working Model -

I/p

↓

Decoder stack

↓

Predict next ~~step~~ token

- Auto regressive text generation is their main purpose
- Kind of generative model
- Used for text generation
- Eg: GPT, Llama

Encoder-Decoder

- Focus on both i/p rep" & o/p generation
- Working Model -

Encoder

↓

Decoder

- Conditional generation (o/p text depend on i/p text) is their main purpose
- Kind of generative model
- Used for translation, summarization, q/aing, etc.
- Eg: T5, FLAN T5
- (Fine tuned Language Net T5)

Chapter-02
Tokens and Embeddings

⇒ **Tokenization :**

Tokenization is a fundamental step in natural language processing & language modelling. It involved breaking down texts into smaller units called tokens (words, sub-words, characters).

⇒ **Why Tokenization is important :**

i) Raw text is not machine readable.
ii) Bridge to embedding.
iii) Vocabulary construction.
iv) Handles linguistic variability.
v) Supports positional encoding.
vi) Context length.

15-09-25

⇒ **Categ. of Tokenization (used in LLMs) :** Eg: Large language Model

1) Word level Tokenization → 'Large' 'language' 'Model'
2) Sub-word level Tokenization → 'La' 'rge' 'lang' 'uage' 'Mo' 'del'
3) Characters level Tokenization → 'L' 'a' 'r' . . . . . 'd' 'e' 'l'
4) Sentence level Tokenization → 'Large language Model'

⇒ **Types of Tokenization**

1) **Byte Pair Encoder (BPE) :**
It splits words into common sub-words units, basically used in GPT models

2) **Word piece Tokenization :**
It is used by Bot, which is similar to BPE but with different rules for spliting.

3) **Sentence piece Tokenization :**
It is used by model like T5, Google, used by also Unigram Language Model.

⇒ **Basic Pipeline of Tokenization to Embedding :**

1. Text Input → large language Model
2. Tokenization → 'Large' 'Language' 'Model'
3. Subword generation → 'La' 'rge' 'Langu' 'age' 'Mo' 'del'
4. Token mapping/vocabulary lookup → 1023, 25, 52 . . 85
5. Special Token → [CLS], [SEP], [PAD], [UKN], . .
6. Embedding layer → $E_{1023}, E_{25}, \ldots E_{85}$

Subword Tokenization:

It is a technique that breaks words into smaller, frequent units called Subwords. This approach helps to handle out of vocabulary words & reduce vocabulary size while preserving semantic structure.

BPE Algorithm:

BPE is a data compression inspired algorithm used for Subword Tokenization. It begins with character level tokens & iteratively merges the most frequent adjacent pairs to form subwords

(1) Pretokenization 1.1 Algorithm starts with a corpus of text data (Raw i/p)
1.2 Corpus is pretokenized to create a list of all unique words. (List of unique words)

(2) Base Vocabulary 2.1: It is initialized with all unique characters found in list of unique words (Const. initial vocab)
2.2 Each word in corpus is then represented as a sequence of symbols from this vocabulary. (Character tokens)

(3) Pair Merging 3.1: Count the frequency of adjacent symbol pairs (Bigram counts) (BiGm)
3.2: Most frequent Bigram is then merged into a symbol & words in the corpus are updated to reflect this merge. (Merging)

This process continues iteratively until the desired vocabulary is reached.

Eg: 1.1: hello
1.2: 'hello'
2.1: 'h' 'e' 'l' 'o'
2.2: 'h' 'e' 'l' 'l' 'o'
3.1: ('h','e') ('e','l') ('l','l') ('l','o') →

BiGm	Freq
he	1
el	1
ll	x2
lo	1

3.2: ('l','l') → 'll' ⇒ 'h' 'e' 'l' 'o' ⇒ 'h' 'e' 'll' 'o'

Eg: aa abc abc
Iteration-1:-
1.1: aa abc abc
1.2: 'aa' 'abc' 'abc'
2.1: 'a' 'b' 'c'
2.2: 'a' 'a' 'a' 'b' 'c' 'a' 'b' 'c'
3.1: ('a','a') ('a','b') ('b','c') ('a','b') ('b','c')
3.2: ('a','b') → 'ab'

BiGm	Freq
aa	1
a,b	2
b,c	2

Iteration-2:-
2.1: 'a', 'b' 'c' 'ab'
2.2: 'a' 'a' 'ab' 'c' 'ab' 'c'
3.1: ('a','a') ('ab','c') ('ab','c')
3.2: ('ab','c') → 'abc'

BiGm	Freq
aa	1
ab,c	2

Iteration-3:-
2.1: 'a', 'b' 'c' 'ab' 'abc'
2.2: 'a' 'a' 'abc' 'abc'
3.1: ('a','a')
3.2: ('a','a') → 'aa'

BiGm	Freq
a,a	1

18-09-25

Q) Consider the following corpus with pre-tokenized text with its freq. Using BPE algo. generate the sub-words & display final vocab

word	Frequency
cat	10
bat	5
bag	12
tag	4
cats	5

Ans: For eg: cat bat bag tag cats
1.1: cat bat bag tag cats
1.2: 'cat' 'bat' 'bag' 'tag' 'cats'
2.1: 'a' 'b' 'c' 'd' 'g' 's' 't'
2.2: 'c' 'a' 't' 'b' 'a' 't' 'b' 'a' 'g' 't' 'a' 'g' 'c' 'a' 't' 's'
3.2: ('c','a') ('a','b') ('t','b') ('b','a') ('a','t') ('t','b') ('b','a') ('a','g') ('g','t') ('t','a') ('a','g') ('g','c') ('c','a') ('a','t') ('t','s')

BiGm	Freq
ca	2
at	3
tb	2
ba	2
ag	2
gt	1
ta	1
bc	1
cs	1

3.2: ('a','t') → 'at'

It-2: 2.1: 'a' 'b' 'c' 'd' 'g' 's' 't' 'at'

2.2: 'c' 'at' 'b' 'at' 'b' 'a' 'g' 't' 'a' 'g' 'c' 'at' 's'

3.1: ('c' 'at') ~~('t' 'at')~~ ('b' 'at') ('b' 'a') ('a' 'g') ('t' 'a') ('a' 'g') ('a' 'at') ('at' 's')

BiGm	Freq
c,at	2
b,at	1
b,a	1
a,g	2
t,a	1
ats	1

3.2: 'c','at' → 'cat'

It-3: 2.1: 'a' 'b' 'c' 'd' 'g' 's' 't' 'at' 'cat'

2.2: 'cat' 'b' 'at' 'b' 'a' 'g' 't' 'a' 'g' 'cat' 's'

3.1: ~~'cat'~~ ('b' 'at') ('b' 'a') ('a' 'g') ('t' 'a') ('a' 'g') ('cat' 's')

BiGm.	Freq
b,at	1
b,a	1
a,g	2
t,a	1
cat,s	1

3.2: 'a','g' → ag

It-1

⇒3.1:

BiGm	Freq
ca	15
at	20
~~tba~~	17
ag	16
ts	1

3.2: 'at' → at

It-2

3.1:

BiGm	Freq
c,at	15
b,at	5
b,a	12
ag	16
t,a	4
at,s	5

2.2: 'a' 'g' → 'ag'

It-3

3.1:

BiGm	Freq
c,at	15
b,at	5
b,ag	12
t,ag	4
~~at~~ s	5

3.2: 'cat' → 'cat'

It-4

3.1:

BiGm	Freq
bat	5
b,ag	12
t,ag	4
cat,s	5

3.2: 'b','ag' → bag

It-4

3.1:

BiGm	Freq
b,at	5
t,ag	4
cat,s	5

3.2: 'bat' → bat

It-5

3.1:

BiGm	Freq
t,ag	4
cat,s	5

3.2: 'cat' 's' → cats

It-6

3.1:

BiGm	Freq
t,ag	4

3.2: 't' 'ag' → tag

9-09-25

2) ⇒ Word piece alg.:

It is a sub-word tokenization algorithm developed by Google, widely used in models like BERT. Its primarily goal is to handle rare or unseen words more effectively than traditional word level tokenization. Instead of treating each word as a token, wordpiece breaks words into smaller, meaningful subword units such as prefix, suffix or roots based on statistical patterns in the training corpus.

(1) Pre tokenization: 1.1: Raw text
1.2: Generate the tokens

(2) Base vocabulary: 2.1: Construction of initial vocabulary
2.2: Generate Character tokens Eg: token → t, ##o, ##k, ##e, ##n

(3) Pair merging: 3.1: Generate Bigram
3.2: Merge x, ##y → xy
##x, ##y → ##xy

score = freq. of pair / freq of 1st token x freq. of 2token

Eg: hello

It-1: 1.1: hello
1.2: 'hello'
2.1: ['h, ##e, ##l, ##o']
2.2: {'h', ##e, ##l, ##l, ##o}
3.1: (h, ##e), (##e, ##l), (##l, ##l), (##l, ##o)
3.2:

pair	score
h, ##e	1/1x2 = 1/2 = 0.5
##e, ##l	1/2x4 = 1/8 = 0.12
##l, ##l	1/4x4 = 1/16 = 0.06
##l, ##o	1/4x1 = 1/4 = 0.25

h, ##e → he

It-2: 2.1: [h, ##e, ##l, ##o, he]
2.2: [he, ##l, ##l, ##o]
3.1: (he, ##l), (##l, ##l), (##l, ##o)
3.2:

pair	score
he, ##l	1/1x4 = 1/4 = 0.25
##l, ##l	1/4x4 = 1/16 = 0.06
##l, ##o	1/4x1 = 1/4 = 0.25

he, ##l → hel

It-3: 2.1: [h, ##e, ##l, ##o, he, hel]
2.2: [hel, ##l, ##o]
3.1: (hel, ##l), (##l, ##o)
3.2:

pair	score
hel, ##l	1/1x2 = 1/2
##l, ##o	1/2x1 = 1/2

hel, ##l → hell

It-4: 2.1: [h, ##e, ##l, ##o, he, hel, hell]
2.2: [hell, ##o]
3.1: (hell, ##o)
3.2:

pair	score
hell, ##o	1/1x1 = 1

hell, ##o → hello

Eg: aa abc abc

It-1: 1.1: aa abc abc
1.2: 'aa' 'abc' 'abc'
2.1: 'a' ##b' ##c
2.2: 'a', ##a, d, ##b, ##e, a, #b, #c
3.1: (a, ##a), (a, ##b), (##b, ##c), (a, ##b) (##b, ##c)
3.2:

pair	score
a, ##a	1/3x1 = 1/3 = 0.33
a, ##b	2/3x4 = 2/12 = 1/6 = 0.16
##b, ##c	2/4x2 = 1/4 = 0.25

a, ##a → aa

It-2: 2-1: [a, ##b, ##c, aa]

2-2: {aa, a, ##b, ##c, a, ##b, ##c}

3-1: (a, ##b), (##b, ##c), (a, ##b), (##b, ##c)

3-2:

pair	score
a ##b	$2/2\times4 = 1/4 = 0.25$
##b, ##c	$2/4\times2 = 1/4 = 0.25$

a, ##b → ab

It-3: 2-1: [a, ##b, ##c, aa, ab].

2-2: {aa, ab, ~~##b~~, ##c, ab, ~~##b~~, ##c}

3-1: (ab, ##c), (ab, ##c)

3-2:

pair	score
ab, ##c	$2/2\times2 = 2/2 = 0.5$

ab, ##c → abc

20-09-25

3) ⇒ Sentence Piece Algorithm:

Note: In Word Piece it isn't possible to find a sub-word in the vocabulary the whole sub-word is tokenized as [UNK] (Unknown Token). Eg: Funnny → F, ##u, ##n, ##n, ##y

Funnny ← [UNK]

whereas in BPE algo, only missing token is replaced by UNK.

3) ⇒ Sentence Piece Algorithm:

It is a language independent sub-word tokenization algorithm developed by Google, designed to train directly on raw text without requiring pre-tokenization.

Unlike Word Piece or BPE, which assume space separated words as input, Sentence Piece treats the input as a raw stream of unicode characters & learns sub-word units using either BPE or Unigram Language Model. This approach enables consistent handling of white-space punctuations & multi-lingual corpora (appln which works on multiple languages).

(1) Initializing the Base vocabulary: It includes all the characters & the most frequent substrings found in the corpus.

(2) Log likelihood loss computation: The algo. calculates log likelihood loss over the entire training data based on the current vocabulary. It aims to find the smallest & the most effective vocabulary that still adequately represents the training data.

$$\boxed{\text{Loss} = \Sigma \text{Freq} * (-\log(P(\text{word})))}$$

(3) Finding candidates for removal: Symbol that contribute less to the overall repr of the data are marked as candidates for removal.

(4) Progressive Vocabulary Pruning: The algo. removes a small percentage of symbols that have the least impact on the log likelihood loss. These are the symbols for which the increase in training loss is the lowest if they are removed. This process is repeated iteratively. After each round of pruning, the log likelihood loss is recalculated with the updated vocabulary & process continues until the vocabulary reaches the desired size.

Q) Consider the following corpus

word	Freq
run	8
bug	5
fun	13
sun	10

Using Sentence Piece algo, construct the vocabulary & display it.

Step 1: Possible word segments - for each word are run → r, u, n, ru, un
bug → b, u, g, bu, ug
fun → f, u, n, fu, un
sun → s, u, n, su, un

Initial vocabulary: [r, u, n, ru, un, b, g, bu, ug, f, fu, s, su]

Compute Freq. of each tokens:

Token	r	u	n	ru	un	b	g	bu	ug	f	fu	s	su
Freq.	8	31	26	3	26	5	5	5	5	13	13	10	10

22-09-25

Q) Let us consider the following corpus [continuation of prev cls]

Word	Freq
run	3
bug	5
fun	13
sun	10

Step 1: Initialize the Base Vocabulary

1.1: Generate the tokens/substrings run → r, u, n, ru, un
bug → b, u, g, bu, ug
fun → f, u, n, fu, un
sun → s, u, n, su, un

1.2: Initialize the vocabulary, InVar → {r, u, n, b, g, ru, un, bu, ug, f, fu, s, su}

1.3: Token frequency

Token	r	u	n	ru	un	b	g	bu	ug	f	fu	s	su	
Freq.	8	31	26	3	26	5	5	5	5	13	13	10	10	Total = 155

1.4: Compute the probability of each token

$$P(t) = \frac{\text{Freq}(t)}{\text{Total Freq}}$$

Token	r	u	n	ru	un	b	g	bu	ug	f	fu	s	su
Prob	0.0193	0.2	0.1677	0.0193	0.1677	0.0322	0.0322	0.0322	0.0322	0.0838	0.0838	0.0645	0.0645

1.5: Find most probable segmentation.

Suppose, run, possible segmentation are (r, u, n) (ru, n) (r, un)

$p(r,u,n) = p(r) * p(u) * p(n) = 0.0193 \times 0.2 \times 0.1677 = \underline{0.00064}$

$p(ru,n) = p(ru) * p(n) = 0.0193 * 0.1677 = \underline{0.00323}$ [max]

$p(r,un) = p(r) * p(un) = 0.0193 \times 0.1677 = \underline{0.00323}$

$p(b,u,g) = p(b) \times p(u) \times p(g) = 0.0322 \times 0.2 \times 0.0322 = \underline{0.00020}$

$p(bu,g) = p(bu) \times p(g) = 0.0322 \times 0.0322 = \underline{0.00103}$ [max]

$p(b,ug) = p(b) \times p(ug) = 0.0322 \times 0.0322 = \underline{0.0013}$

25-09-25

$p(f,u,n) = p(f) \times p(u) \times p(n) = 0.0838 \times 0.2 \times 0.1677 = 0.00281$
$p(fu,n) = p(fu) \times p(n) = 0.0838 \times 0.1677 = 0.01405$ [max]
$p(f,un) = p(f) \times p(un) = 0.0838 \times 0.1677 = 0.01405$

$p(s,u,n) = p(s) \times p(u) \times p(n) = 0.0645 \times 0.2 \times 0.1677 = 0.00216$
$p(su,n) = p(su) \times p(n) = 0.0645 \times 0.1677 = 0.01081$ [max]
$p(s,un) = p(s) \times p(un) = 0.0645 \times 0.1677 = 0.01081$

word	Freq.	Split/most probable	Score
run	3	(ru,n)	0.00325338
bug	5	(bu,g)	0.00103
fun	13	(fu,n)	0.01405
sun	10	(su,n)	0.01081

Step2: Calculate Log Likelihood loss.

$loss = \sum Freq * (-\log(p(split)))$
$= 3 * (-\log(0.00325338)) + 5 * (-\log(0.00103)) + 13 * (-\log(0.01405)) + 10 * (-\log(0.01081))$
$= 66.102.$

Step3: Removal of candidate from vocabulary.

To determine which token to remove, we need to calculate the associated loss for each token in the vocabulary. However, that is not an elementary token, then compare the loss.

Suppose, you want to eliminate 'un'.

St-1: Var = {r, u, n, ru, b, bu, ug, f, fu, s, su}

After eliminating 'un', again loss value will be 66.102.

27-09-25 Suppose, you want to eliminate 'ru'.

Var = {r, u, n, un, b, bu, ug, f, fu, s, su}

After eliminating 'ru', again loss value will be 66.102.

Substring	Ass: loss
ru	66.102
un	66.102
bu	66.102
ug	66.102
fu	66.102
su	66.102

As, all Associated loss are same, pick anyone for removal

Let as assume, we removed un.

St-2: 1.2: Var = {r, u, n, ru, b, bu, ug, f, fu, s, su}

1.3:

Token	...	Everything except 'un'
Freq	...	Total=129

1.4: Similarly without 'un'.

Differentiate between BPE, WP & SP:

Features	BPE	WP	SP
Core idea/concept	It uses greedy merging of frequent char. pairs.	It uses likelihood based sub-word merge.	It uses unigram lang. model or BPE over raw text.
Training i/p	It uses pre-tokenized text.	It uses pre-tokenized text.	It uses raw text (no pre processing needed)
Sampling capacity	It is deterministic.	It is deterministic.	It can sample multiple tokenizn.
Multi-lingual support	Limited.	Limited.	Strongly supports.
Used in LLM (Eg)	GPT with Falcon	BERT, DistilBERT, ALBERT	T5, XLNet, mT5, PaLM. (developed by Google)

6-10-25

Word/Token embedding:

- Static
  - Word2vec
    - CBow
    - Skip gram
  - Glove
- Contextual

- CBow (Continuous Bag of Word): It predicts the current word based on surrounding context words.
- Skip gram: It predicts surrounding content word by given current word.

Eg: Generate a dense vector representation for following sentence using CBow model.
Sentence: The cat sits on the mat
Target word: sits
Context window: 2

Step 1: Construct a vocabulary: {the, cats, sits, on, mat}
Step 2: One-hot vector representation - the : [1,0,0,0,0]
cat : [0,1,0,0,0]
sits : [0,0,1,0,0]
on : [0,0,0,1,0]
mat : [0,0,0,0,1]

Step 3: Consider content words (the, cat, on, the) to get a combined i/p of CBow that averages one-hot vectors

Avg-vector = $\frac{1}{4}\left(\underbrace{[1,0,0,0,0]}_{\text{the}} + \underbrace{[0,1,0,0,0]}_{\text{cat}} + \underbrace{[0,0,0,1,0]}_{\text{on}} + \underbrace{[1,0,0,0,0]}_{\text{the}}\right)$ =

= $\frac{1+0+0+1}{4}, \frac{0+1+0+0}{4}, 0, 0.25, 0$ = [0.5, 0.25, 0, 0.25, 0]

Step 4: Initialize the weight matrix

Let $W =$

	dim1	dim2	dim3	dim4	dim5
the	0.1	0.2	0.3	0.4	0.5
cat	0.6	0.7	0.8	0.9	0.1
sit	0.2	0.3	0.4	0.5	0.6
on	0.1	0.8	0.9	0.1	0.2
mat	0.3	0.4	0.5	0.6	0.7

Hidden Layer

Step 5: Hidden layer o/p : By multiplying Avg vector with weight matrix (Avg vector × W)

$= [0.6, 0.25, 0, 0.25, 0] \times [0.1, 0.2, 0.3, 0.1, 0.5]$

$= [0.375, 0.525, 0.725, 0.525, 0.3]$

Step 6: Initialize weight matrix for o/p layers.

Let $W' =$

$$\begin{bmatrix} 0.8 & 0.4 & 0.2 & 0.1 & 0.5 \\ 0.3 & 0.6 & 0.9 & 0.8 & 0.2 \\ 0.4 & 0.5 & 0.7 & 0.3 & 0.1 \\ 0.9 & 0.1 & 0.2 & 0.6 & 0.8 \\ 0.5 & 0.7 & 0.3 & 0.4 & 0.6 \end{bmatrix}$$

Step 7: O/p layer calculation : Hidden layer o/p × W'.

$\Rightarrow$ Output (o/p layer) $= [1.4325, 0.9525, 1.4175, 0.9125, 0.7625]$

Step 8: Softmax Activation

To apply softmax, need to exponentiate

exp_store $= [\exp(1.4325), \exp(0.9525), \ldots]$

$= [4.1903, 2.5926, 4.1239, 2.4906, 2.1435]$

sum_exp_stores $= 4.1903 + 2.5926 + 4.1239 + 2.4906 + 2.1435$

$= 15.5409$

Step 9: Predict the probability,

$$P = \left[\frac{4.1903}{15.5409}, \frac{2.5926}{15.5409}, \frac{4.1239}{15.5409}, \frac{2.4906}{15.5409}, \frac{2.1435}{15.5409}\right]$$

the = 0.2697
cat = 0.1668
sit = 0.2653
on = 0.1603
mat = 0.1379

Chapter-03

Looking inside LLMs

09-10-25

→ Transformer:

- A transformer is a type of AI model that learns to understand & generate human-like text by analyzing patterns in large amounts of text data.
- It process whole sentence in parallel.
- Transformer architecture uses self attention to transform one whole sentence into a single sentence.
- This is useful where older models work step by step & it helps to overcome the challenges seen in the models like RNN, LSTM (Long-short term memory)
- Traditional models like RNN suffers from the vanishing gradient problem which leads to long term memory loss & RNN process takes one word at a time sequentially.

Eg: "In the sentence "XYZ went to France in 2019 when there were no cases of covid & there he met the President of that country."

Here that country refers to France. However RNN struggle to link that country to France since it processes each word in sequence leading to loosing context over long sentences. This limitation prevents RNN from understanding the full meaning of that sentence.

- While adding more memory cells in LSTM that help to address the vanishing gradient issues but still process words one by one.
- This sequential process means LSTM cannot understand or analyze an entire sentence at once.
- Transformers were inspired by Encoder-Decoder architecture found in RNN However, instead of recurrence transformer model is completely based on attention mechanism.
- The model doesn't generate the text all in one operation rather it generates one token at a time. After each token generation, the input prompt for the next generation step by appending the output token to the end of input prompt that means the model consumes the earlier predictions to make later prediction, are called Auto regressive models.

10-10-25

- The method of choosing a single token from the probability is called the decoding strategy.
- The easiest decoding strategy to pick the token with the highest probability score. And choosing the highest scoring token everytime is called greedy decoding.

Transformer is made up of two successive components -

(i) Attention layer i.e, it is mainly concerned with incorporending with relevent information from other input tokens & positions.

(ii) The fit forward neural - token representation individually, helping the model to capture complex patterns. After attention has modelled relationship between tokens.

→ Attention Mechanism is a technique used in machine learning & AI to improve the performance of the model.
There are two steps involved in attention mechanisms -
(i) A way to score how relevant input of the previous token into the current token being processes.
(ii) Using those pores combine the information from various position

→ Advantages : (Attention Mechanism)
- It enhance the performance of neural machine translate system.
- Ability to address the bottle neck problem where all the information from a long input sequence needs to be compressed into a single fixed length vector.
- With attention the decoder can analyze all the encoder's hidden state with attention & dynamically weight their importance.
- It also helps to reduce the managing gradient problem by connecting all encoders to decoders.

11-10-25

Q) Consider translating the english phrase "I love bananas" into Spanish as "amo las bananas" The hidden states of the encoders are $e_n = [0.1, 0.2] [0.3, 0.8] [0.5, 0.6]$. Compute the attention scores for the decoder, which outputs the spanish word 'las' having a hidden state of $[0.4, 0.6]$.

The dot product (Attention score) i.e similarity between Encoder's hidden state & Decoder's hidden state) are computed as:

I = $0.1 \times 0.4 + 0.2 \times 0.6 = 0.04 + 0.12 = \underline{0.16}$.
love = $0.3 \times 0.4 + 0.8 \times 0.6 = 0.12 + 0.48 = \underline{0.6}$
bananas = $0.5 \times 0.4 + 0.6 \times 0.6 = 0.2 + 0.36 = \underline{0.56}$

- As similarity between different hidden states are not directly comparable, therefore normalise these score to convert the similarity score to probabilistics score by using softmax function.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

$e^{0.16} = 1.17$
$e^{0.6} = 1.82$
$e^{0.56} = 1.75$

$$\text{Softmax}(I) = \frac{e^{0.16}}{e^{0.16} + e^{0.6} + e^{0.56}} = 0.25$$

$$\text{Softmax}(\text{love}) = \frac{e^{0.6}}{e^{0.16} + e^{0.6} + e^{0.56}} = 0.38$$

$$\text{Softmax}(\text{bananas}) = \frac{e^{0.56}}{e^{0.16} + e^{0.6} + e^{0.56}} = 0.37$$

- The Attention dist"/output i.e, $\alpha(t)$ gives us the fixed length input context vector for the decoder state dt are as follows:

$$\alpha(t) = \sum_{i=1}^{t} \text{softmax}(t) \cdot e^{(t)}$$

$= 0.25 \times [0.1, 0.2] + 0.38 [0.3, 0.8] + 0.37 [0.5, 0.6]$
$= [0.25 \times 0.1, 0.25 \times 0.2] + \ldots$
$= [0.025, 0.05] + [0.114, 0.304], [0.185, 0.222]$
$= [0.324, 0.576]$.

Final decoder hidden state for las, after concatenation with initial decoder hidden state, respectively as $= [0.32, 0.51, 0.4, 0.6]$

Self Attention:

→ It is a mechanism to capture dependencies & relationships within input sequences. It allows the model to identify & weight the importance of different parts of the i/p sequence by attending itself.

Why Self Attention is required/important:

→ i) Long range dependencies: It allows the model to capture relationship between distant element in a sequence.

ii) Contextual understanding: It helps the model to understand the context & assign appropriate weights to each element based on its relevance.

iii) Parallel computation: It can be computed in parallel for each element, parallel in sequence.

13-10-25

How Self attention works?

S-1: The 1st step is to generate the query (Q), key (K) & Value (V). $\begin{bmatrix} Q = X \cdot W_Q \\ K = X \cdot W_K \\ V = X \cdot W_V \end{bmatrix}$

S-2: Calculate attention score = QK^T

S-3: Scale the scores = $\frac{QK^T}{\sqrt{d_k}}$

S-4: Apply the softmax function to calculate attention weight = $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$

S-5: Calculate final output (O) = Attention weight × V

Eg (Single head without mask): "the cat sat". Consider the sentence, compute the self attention representation for the word 'cat'.

S-1: Let embedding size, $d_{model} = 3$

head size, $d_k = d_v = 2$

Consider, $X = \begin{matrix} \\ \text{the} \\ \text{cat} \\ \text{sat} \end{matrix} \begin{matrix} \text{dim1} \\ \begin{bmatrix} 0.2 & 0.5 & 0.1 \\ 0.9 & 0.1 & 0.3 \\ 0.4 & 0.7 & 0.8 \end{bmatrix} \end{matrix} \begin{matrix} \text{dim2} \\ \\ \\ \\ \end{matrix} \begin{matrix} \text{dim3} \\ \\ \\ \\ \end{matrix}$

$W_Q = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.7 \\ 0.5 & 0.6 \end{bmatrix}$ $W_K = \begin{bmatrix} 0.4 & 0.8 \\ 0.9 & 0.1 \\ 0.3 & 0.5 \end{bmatrix}$ $W_V = \begin{bmatrix} 0.6 & 0.2 \\ 0.1 & 0.9 \\ 0.7 & 0.3 \end{bmatrix}$

$Q = XW_Q = \begin{bmatrix} 0.2 & 0.5 & 0.1 \\ 0.9 & 0.1 & 0.3 \\ 0.4 & 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.7 \\ 0.5 & 0.6 \end{bmatrix} = \begin{bmatrix} 0.17 & 0.47 \\ 0.26 & 0.52 \\ 0.58 & 1.09 \end{bmatrix}$

$K = XW_K = \begin{bmatrix} 0.2 & 0.5 & 0.1 \\ 0.9 & 0.1 & 0.3 \\ 0.4 & 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 0.4 & 0.8 \\ 0.9 & 0.1 \\ 0.3 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.56 & 0.26 \\ 0.54 & 0.88 \\ 1.03 & 0.79 \end{bmatrix}$

$V = X \cdot W_V = \begin{bmatrix} 0.2 & 0.5 & 0.1 \\ 0.9 & 0.1 & 0.3 \\ 0.4 & 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 0.6 & 0.2 \\ 0.1 & 0.9 \\ 0.7 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.24 & 0.52 \\ 0.76 & 0.36 \\ 0.87 & 0.95 \end{bmatrix}$

S-2: Attention Score ⇒ QK^T

$$\begin{bmatrix} 0.17 & 0.47 \\ 0.26 & 0.52 \\ 0.58 & 1.09 \end{bmatrix} \cdot \begin{bmatrix} 0.56 & 0.54 & 1.03 \\ 0.26 & 0.88 & 0.79 \end{bmatrix} = \begin{bmatrix} 0.2174 & 0.5054 & 0.5469 \\ 0.2808 & 0.598 & 0.6786 \\ 0.6082 & 1.2724 & 1.4585 \end{bmatrix}$$

S-3: $\text{score} = \frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} 0.1537 & 0.3573 & 0.3863 \\ 0.1985 & 0.4228 & 0.4798 \\ 0.43 & 0.8997 & 1.0313 \end{bmatrix}$

S-4: Attention Weight = $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ = $\begin{matrix} \\ \text{the} \\ \text{cat} \\ \text{cat} \end{matrix} \begin{bmatrix} x & y & z \\ - & - & - \\ - & - & - \end{bmatrix}$ $x = \frac{e^x}{e^x+e^y+e^z}$, $y = \frac{e^y}{e^x+e^y+e^z}$

$$= \begin{bmatrix} 0.2867 & 0.3514 & 0.3618 \\ 0.2796 & 0.3499 & 0.3704 \\ 0.2260 & 0.3615 & 0.4124 \end{bmatrix} \begin{matrix} \leftarrow \text{softmax(the)} \\ \leftarrow \text{softmax(cat)} \\ \leftarrow \text{softmax(cat)} \end{matrix}$$

S-5: Output (O) = Attention weight × V

$$= \begin{bmatrix} 0.2867 & 0.3514 & 0.3618 \\ 0.2796 & 0.3499 & 0.3704 \\ 0.2260 & 0.3615 & 0.4124 \end{bmatrix} \times \begin{bmatrix} 0.24 & 0.52 \\ 0.76 & 0.36 \\ 0.87 & 0.95 \end{bmatrix} = \begin{bmatrix} 0.6506 & 0.6192 \\ 0.6552 & 0.6232 \\ 0.6877 & 0.6399 \end{bmatrix}$$

Self Attention representation for word cat is $\begin{bmatrix} 0.6552 & 0.6232 \end{bmatrix}$.

16-10-25

⇒ Multi-head Attention:

- It involves repeating the self-attention process multiple times, with each repeatation using different linear projections of the input data.
- Using multiple attention heads allow the model to capture a reacher set of dependencies in the input sequence.

Eg: One head might focus on the overall sentence structure, while another zooms in on specific details, ~~while another zooms in on specific details~~.

- By combining diverse perspectives, multi-head attention provides a more comprehensive understanding of the input, much like how humans consider multiple aspects of information simultaneously.

→ Algorithmic Step (or) Work Flow of Multi-head Attention:

S-1: Linear Projection: For each attention head the input sequence is linearly projected into separate queries (Q), Keys (K), Values (V) using distinct learn weight matrices.

$[Q = XW_{Q_i} \quad K = XW_{K_i}, \quad V = XW_{V_i}]$

S-2: Attention Calculation per head (or) Scaled Dot Product Attention.

$$\text{Attention}_i(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

- Attention score = $Q_i K^T$
- Scale the score = $\frac{QK^T}{\sqrt{d_k}}$
- Apply softmax = $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$

- Att. wt = $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$

S-3: After processing through the individual attention head, the outputs of all heads are concatenated together i.e.

$$\text{Concatenated o/p} = \text{Concat}(\text{attention}_1, \text{attention}_2, \ldots \text{attention}_n)$$

S-4: Final linear Projection: The concatenated output is passed through another linear projection using a weight matrix to produce the Final Output (FO).

$$\text{FO} = \text{Concatenated o/p} \times W_0$$

Eg (Multi-head Attention): "the cat sat". Consider the sentence & compute the attention score

S-1: Consider the model with $d_{model} = 4$, no. of heads = 2, $d_k = d_v = \frac{d_{model}}{heads} = \frac{4}{2} = 2$.

$$X = \begin{matrix} \\ \text{the} \\ \text{cat} \\ \text{sat} \end{matrix} \begin{matrix} \begin{matrix} \text{dim1} & \text{dim2} & \text{dim3} & \text{dim4} \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix} \qquad W_Q = W_K = W_V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$Q = XW_Q = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$K = XW_K = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$V = XW_V = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

S-2: Attention score = QK^T (For head 1)

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{Scale the score} = \frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} 0.7071 & 0 & 0 \\ 0 & 0.7071 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{Apply softmax} = \begin{bmatrix} 0.5034 & 0.2482 & 0.2482 \\ 0.2482 & 0.5034 & 0.2482 \\ 0.3333 & 0.3333 & 0.3333 \end{bmatrix}$$

$e^{0.7071} = 2028$

$$\text{Attention weight} = \begin{bmatrix} 0.5034 & 0.2482 & 0.2482 \\ 0.2482 & 0.5034 & 0.2482 \\ 0.3333 & 0.3333 & 0.3333 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.5034 & 0.2482 \\ 0.2482 & 0.5034 \\ 0.3333 & 0.3333 \end{bmatrix}$$

17-10-25

For head 2, $X = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ $\quad W_Q = W_K = W_V = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$

$Q = XW_Q$ $[Q = K = V = XW_Q = XW_k = XW_v$

$$= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$QK^T = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

$$\frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} 0.7071 & 0.7071 & 0.7071 \\ 0.7071 & 0.7071 & 0.7071 \\ 0.7071 & 0.7071 & 1.4142 \end{bmatrix}$$

Apply softmax - $\begin{bmatrix} 0.3333 & 0.3333 & 0.3333 \\ 0.3333 & 0.3333 & 0.3333 \\ 0.2482 & 0.2482 & 0.5034 \end{bmatrix}$

Attention weight = $\begin{bmatrix} 0.3333 & 0.3333 & 0.3333 \\ 0.3333 & 0.3333 & 0.3333 \\ 0.2482 & 0.2482 & 0.5034 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.3333 & 0.9999 \\ 0.3333 & 0.9999 \\ 0.5034 & 0.9998 \end{bmatrix}$

S-3: Head Concatenation

Concatenated o/p = $\begin{bmatrix} 0.5034 & 0.2482 & 0.3333 & 0.9999 \\ 0.2482 & 0.5034 & 0.3333 & 0.9999 \\ 0.3333 & 0.3333 & 0.5034 & 0.9998 \end{bmatrix}$

S-4: Final Output = Concatenated o/p $\times W_0$ (Generate FO)

$$= \begin{bmatrix} 0.5034 & 0.2482 & 0.3333 & 0.9999 \\ 0.2482 & 0.5034 & 0.3333 & 0.9999 \\ 0.3333 & 0.3333 & 0.5034 & 0.9998 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5034 & 0.2482 & 0.3333 \\ 0.2482 & 0.5034 & 0.3333 \\ 0.3333 & 0.3333 & 0.5034 \end{bmatrix}$$

23-10-25

→ Differentiate between Single Head, Multi Head & Masked Attention :

Single-Head	Multi-Head	Masked Attention
i) It uses single-attention mechanism to compute attention weight & generate a context vector.	i) It uses multiple attention mechanism in parallel, each computing attention weight & generating a context vector.	i) It is used in Auto-regressive model where future are masked to prevent the model from attending to them.
ii) It may struggle to capture complex relationships, between input elements.	ii) It captures more complex relationship between input element by combining the outputs of multiple attention heads.	ii) It is used to prevent attending to future tokens while single & multi head attention do not use masking by-default.
iii) Only one set of query, key & value weight matrices is used.	iii) Multiple attention heads use separate set of query, key & value weight matrices & they operate in parallel & their o/ps are concatinated at the end.	iii) It is same as multi-head attention but before applying Softmax, a mask is added that sets scores of future position to $-\infty$.
iv) Attention $(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V$	iv) Multihead $(Q,K,V) = \text{Concat}(h_1, h_2 \cdots h_n) \cdot W_o$	iv) Masked attention $(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \text{mask}\right) \times V$
v) Eg: In the sentence, "The cat sat on the mat", Single-head attention might focus mainly on Subject + verb i.e, cat-sat but miss out "on-mat" relationship.	v) Eg: In this case, One head might focus on Subject+verb i.e, cat-sat relationship, another head focused on prepositions like 'on-mat', another might learn co-references like "The-cat".	

Q) Why the Mask is needed ?

- In Masked Attention, the goal is to prevent a word from 'seeing' future words during training / generation.
- This is important because the model must learn to predict the next token only from the tokens, it has already seen but not by looking ahead at future tokens.

eg. "The cat sat on the mat"

Step	Current Token being generated	Allowed to attend	Masked effect
1	the	only itself	nothing to be masked yet
2	cat	[the, cat]	→ can't look at 'sat', 'on', 'the', 'mat'
3	sat	[the, cat, sat]	→ can't look at 'on', 'the', 'mat'
4	on	[the, cat, sat, on]	→ can't look at 'the', 'mat'
5	the	[the, cat, sat, on, the]	→ can't look at 'mat'
6	mat	[the, cat, sat, on, the, mat]	Final

⇒ Mathematical Representation of Masked Attention :

	the	cat	sat	on	the	mat
the	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
cat	0	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$
sat	0	0	0	$-\infty$	$-\infty$	$-\infty$
on	0	0	0	0	$-\infty$	$-\infty$
the	0	0	0	0	0	$-\infty$
mat	0	0	0	0	0	0

0 → Visible

$-\infty$ → Not Visible

24-10-25

Chapter-04 Part-2

Text Classification

→ Differentiate Text Classification with Representation type model & Generative type model:

Representation type Model

i) It uses models primarily designed to create a numerical (meaningful) representation of i/p text. These representation/embedding capture the semantic meaning & content of text

ii) In this model, a encoder is used to generate contextualised embedding vector & then pass through a task specific layer also called classification head to map numerical vector to final class probability.

iii) To find the optimal decision boundary in the high dimensional embedding space that cleanly separate the classes, this makes them discriminative model. (GOAL)

iv) Eg: i/p: "Congratulations, You won a free IPhone."
o/p: Spam

v) Models Used: BERT, RoBERTa

Generative type Model

i) It uses models whose original training objective is text generation. To perform classification their generative capability is repurposed using prompting.

ii) In this model, the classification instruction is embedded directly into the input text, a process called prompt engineering.

iii) To generate a textual response, that fulfills the instruction in the prompt. (GOAL)

iv) Eg: i/p: "Congratulations, You won a free IPhone"
o/p: This msg looks like spam or fishing attempt.

v) Models Used: GPT, T5

25-10-25

→ Text Classification with Representation Models:

Task Specific Model (label)

i) It is trained to perform a particular NLP task such as Sentiment Analysis, Text classification, Summarization, etc. It is fully trained or fine-tuned on label data for that specific task.

ii) The main purpose is to directly output the final result or decision for the specific appln

iii) The goal is to perform a defined task.

iv) It is optimized for one task.

Embedding Model (vector)

i) It is trained to produce dense-vector repr" for words, sentence or documents capturing semantic meanings.

ii) The main purpose is to convert text into numerical form that can be reused for many tasks.

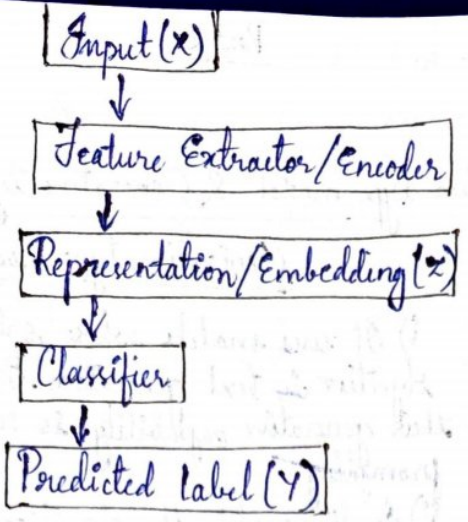
iii) The goal is to represent meaning in vector form

iv) It is reusable for many task.

→ Supervised Classification in Representation Models:

It refers to a process of training a model to map input data to meaningful representation (features/embeddings) & then assigning class labels based on those representations.

Architecture:



Eg: Image of a dog

27-10-25

Q) What if we do not have a label data?

A: If we do not have label data we can't train a supervised classifier directly rather use self supervised, unsupervised or transfer learning to train or reuse a representation model without labels and optionally add a small amount of labels or weak labels for final classification.

1 Solution to this problem is - use of zero-shot model.

- one-shot model
- few-shot model

→ How Zero-shot model works?

- Most ~~modern~~ zero-shot model is a model that can make prediction on new classes/tasks without having been explicitely trainned on them.
- Most modern zero-shot models (eg. clip) [one-shot model] (eg. Flamingo) [few-shot model] are trainned on very large dataset of images & texts together. (eg, an img of a dog & the caption - The dog is running).
- During training they learn to align image features & text features in the same embedding space, this allows the model to recognise classes without seeing labeled trainning data.

⇒ Step 1: It encodes the image into an image embedding.
Step 2: It encodes the text label into text embedding.
Step 3: It computes cosine similarity between image & text embedding.
Step 4: It selects closest text as the predicted class.

→ How cosine similarity between image & text embedding works?

$$\text{cosine similarity}(A,B) = \frac{A \cdot B}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{j=1}^{n} B_j^2}}$$

Eg: Consider the following images & calculate similarity among them to provide label info.
Image 1 (dog) i.e, A = [0.2, 0.5, 0.8, 0.1]
Image 2 (another dog), i.e, B = [0.3, 0.4, 0.7, 0.2]
Image 3 (cat), i.e, C = [0.9, 0.1, 0.2, 0.3]

$$\text{cosine sim}(A,B) = \frac{0.2\times0.3 + 0.5\times0.4 + 0.8\times0.7 + 0.1\times0.2}{\sqrt{0.2^2+0.5^2+0.8^2+0.1^2}\times\sqrt{0.3^2+0.4^2+0.7^2+0.2^2}} = \frac{0.84}{0.96\times0.8839} = 0.98 = \underline{98\%}$$

$$\text{cosine sim}(A,C) = \frac{0.42}{0.9695} = 0.45 \approx \underline{45\%}$$

$$\text{cosine sim}(B,C) = \frac{0.51}{0.88\times0.97} = 0.59 = \underline{59\%}$$

30-10-25

Chapter-05

Text clustering & Topic modelling

Text clustering & Topic modelling are unsupervised analysis technique that aims to discover structure & pattern within large collection of documents while they share the same goal i.e grouping or organizing text data but they differ in their approach, representation & interpretation.

⇒ Difference between Text Clustering & Topic Modelling :

Text Clustering	Topic Modelling
i) It aims to group similar text based on their semantic content, meaning & relationship.	i) It refers to process of representing, generating or understanding text using statistical / neural models
ii) The goal is to group similar text.	ii) The goal is to represent/generate text patterns.
iii) It is a distance / similarity based approach that groups documents into cluster.	iii) It is a probabilistic approach that assumes each document is a mixture of multiple topics & each topic is a distribution of words & it covers latent (hidden) semantic structure.
iv) It is usually a hard clustering method where each document belongs to one cluster.	iv) It can be viewed as a soft clustering method where document belongs to multiple topics with varying probabilities.

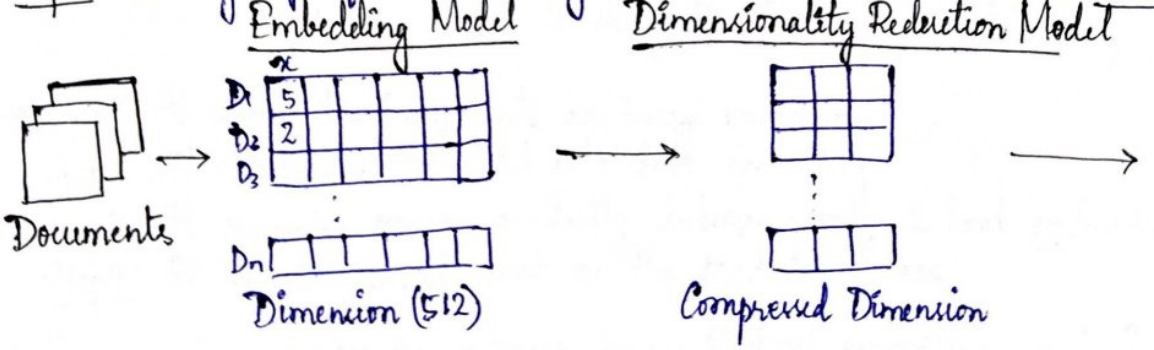
⇒ A common pipeline for Text Clustering :

There are 3 steps in common pipeline for Text Clustering :

Step 1 : Convert the input documents to embedding with an embedding model.

Step 2 : Reduce the dimensionality of embeddings with a dimensionality reduction model.

Step 3 : Find groups of semantically similar documents with a cluster model.



⇒ Dimensionality Reduction Model

As the no. of dimensions increases there is an exponential growth in no. of possible values within each dimension. Finding all sub-spaces within each dimension becomes increasingly complex, as a result it is difficult to identify meaningful Clusters.

11.25

Uniform Manifold Approximation & Projection (UMAP):

- UMAP is a non-linear dimensionality Reduction technique that helps to represent high dimensional data in a lower dimensional space while preserving the structure & relationship in the data as much as possible. For mathematical transformation it uses non-linear optimisation, graph based or distance preserving method.
- The main core mechanism of UMAP is to built a graph representation of data in high dimensions & then optimizes a low dimensional projection that maintains both -
(a) local structure - nearby points stay close.
(b) Global structure - overall data distribution remains meaningful.
- UMAP works in two phases -
(a) Constructing a high dimensional graph.
(b) Optimizing the lower dimensional embeddings.

Eg: Consider the following 6 points in 3D to form two clusters where each point A, B, C, D, E, F represents a data sample in HD space.

Points	x	y	z
A	0.0	0.0	0.0
B	0.2	0.1	0.0
C	0.1	-0.2	0.1
D	3.0	3.1	3.0
E	3.2	2.9	3.1
F	2.9	3.2	3.2

Step 1: Compute Euclidean distance

$$D(A,B) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + \dots}$$

Points	A	B	C	D	E	F
A	0	0.2236	0.2449	5.2545	5.3160	5.3749
B	0.2236	0	0.3317	5.0833	5.1429	5.2096
C	0.2449	0.3317	0	5.2640	5.3122	5.3860
D	5.2545	5.0833	5.2640	0	0.3	0.2449
E	5.3160	5.1429	5.3122	0.3	0	0.4358
F	5.3749	5.2096	5.3860	0.2449	0.4358	0

Step 2: Find k-Nearest Neighbour where k = 2.

Points	1st Nearest	2nd Nearest
A	B (0.2236)	C (0.2449)
B	A (0.2236)	C (0.3317)
C	A (0.2449)	B (0.3317)
D	F (0.2449)	E (0.3)
E	D (0.3)	F (0.4358)
F	D (0.2449)	E (0.4358)

03-11-25

Step3: Compute local parameters to check local connectivity strength

UMAP uses two parameters for each point:

i) ρ (rho) - that measures the distance to the nearest neighbour.

ii) σ (sigma) - scaling factor to make total local probabilities i.e $\sigma = 0.5$ (Here).

To compute the local connectivity strength i.e affinity from i to j is

$$a_{ij} = \begin{cases} 1 & \text{if } d_{ij} \leq \rho_i \\ e^{-\left(\frac{(d_{ij} - \rho_i)}{\sigma_i}\right)} & \text{if } d_{ij} > \rho_i \end{cases}$$

For Cluster-1: For A, $\rho_A = 0.2236$

For A→B, $a_{AB} = 1$ ($0.2236 (\rho_i) \geq 0.2236$ (A to B) So, $a_{ij} = 1$)

For A→C, $a_{AC} = e^{-\left(\frac{(0.2449 - 0.2236)}{0.5}\right)} = 0.961$ ($0.2449 > \rho_A$ [i.e 0.2236])

For B, $\rho_B = 0.2236$

For B→A, $a_{BA} = 1$ ($d = 0.22$)

For B→C, $a_{BC} = e^{-(0.11/0.5)} = 0.802$ ($d = 0.33$)

For C, $\rho_C = 0.2449$

For C→A, $a_{CA} = 1$ ($d = 0.24$)

For C→B, $a_{CB} = e^{-(0.09/0.5)} = 0.836$ ($d = 0.33$)

06-11-25

For Cluster-2: For D, $\rho_D = 0.2449$

For D→E, $a_{DE} = 1$

For D→E, $a_{DF} = e^{-\left(\frac{(0.3 - 0.24149)}{0.5}\right)} = 0.895$

For E, $\rho_E = 0.3$

For E→D, $a_{ED} = 1$

For E→F, $a_{EF} = e^{-\left(\frac{(0.4358 - 0.3)}{0.5}\right)} = 0.7621$

For F, $\rho_F = 0.2449$

For F→D, $a_{FD} = 1$

For F→E, $a_{FE} = e^{-\left(\frac{(0.4358 - 0.2449)}{0.5}\right)} = 0.682$

Step4: Symmetrize the Affinities

UMAP symmetrizes directed affinities to get an undirected fuzzy graph using:

$$w_{ij} = a_{ij} + a_{ji} - a_{ij} \times a_{ji}$$

For Cluster 1:

Pair	Formula
A-B	$1+1-1\times1 = 1.00$
A-C	$0.961+1-0.961\times1 = 1.00$
B-C	$0.802+0.836-0.802\times0.836 = 1.638-0.671 = 0.967$

24-11-25

Chapter-06

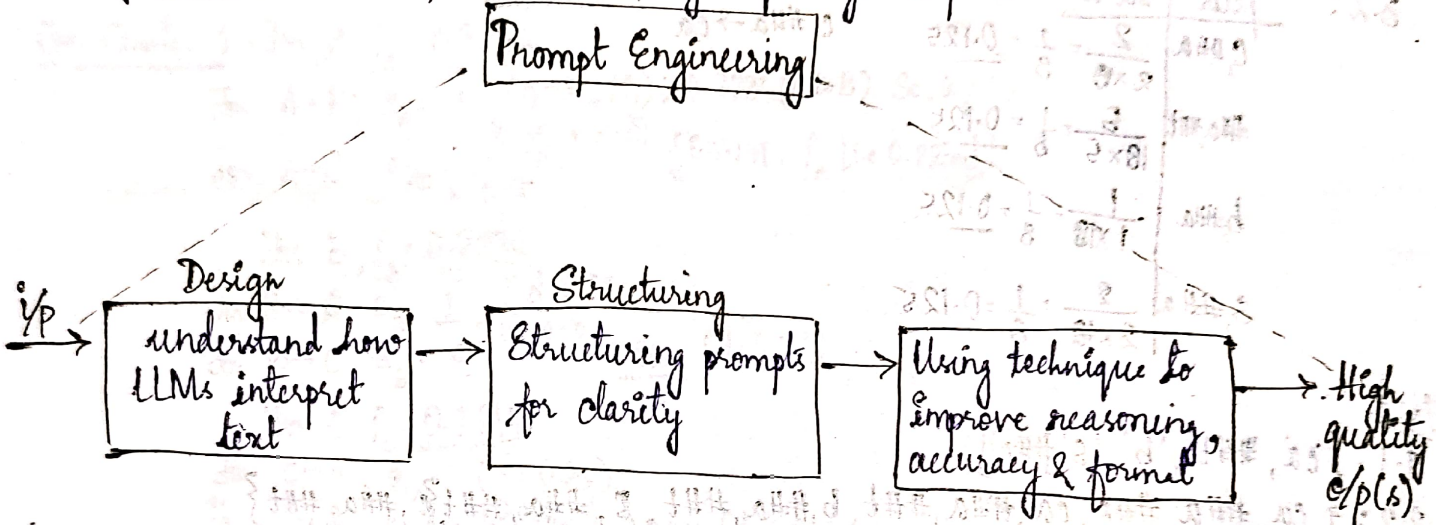
Prompt Engineering

⇒ **What is prompt?**

A prompt is an input that user ~~to~~ give to an AI model to make it generate a response

⇒ **Prompt Engineering:**

It is a process of designing, structuring & optimizing prompts to LLM so that the model produces accurate, relevant & high quality outputs.



⇒ **Why Prompt Engineering is important?**

i) **For Better output:** Good prompts lead to more accurate, relevant & useful responses.

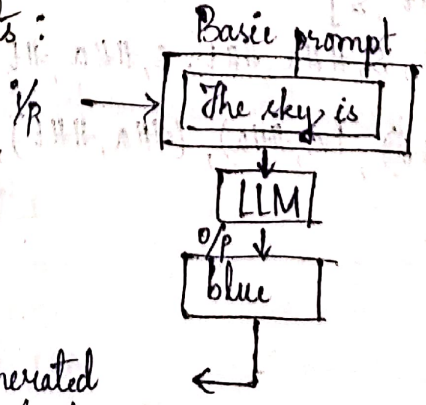
ii) **Clearer communication:** It helps the model to understand what you want, by reducing misunderstandings.

iii) **Efficient interaction:** Effective prompts save time & efforts, i.e, to get the information that you need faster.

iv) **Unlocking capabilities:** It can tap into the model's full potential, revealing hidden strengths & abilities.

v) **Reducing error:** Thoughtful prompts minimize mistakes, biases & off-topic responses

⇒ **Components of an effective prompts:**



1) Introduction: Tells the model exactly what you want it to do.
2) Content: Give any background, constraints that shape the answer.
3) Input data: Provide the text of data the model should work on.
4) Output Format: Specify how do you want the results.
5) Example: A quick help to the model match the style hereafter.

27-11-25

Q) How to control model's output?

A: To control the model's output, adjust the model's parameter like Temperature, Top k, Top P.

Temperature: It controls random or deterministic the model's next token.
- It controls the degree of randomness in token selection. It defines how likely it is to choose tokens that are less probable.
- Lower temperatures are good for prompts that expect a more deterministic response, while higher temperatures can lead to more diverse or unexpected results.
- Use a low temperature for factual answers while a higher one for stories or brainstorming.

Top k: It controls the top k probable tokens before sampling.
- Top k selects the top k most likely tokens from the model's predicted distribution.
- The higher top k is the more creative & varied model's output while the lower top k is more restrictive & fractual the model's output.
- A top k of one is equivalent to greedy decoding.

Top P: It controls to choose tokens from the smallest set whose cumulative probability ≥ P
- Top P sampling selects the top tokens whose cumulative probability doesn't exceed a ~~short~~ certain value P.
- Instead of a fixed K, set a cumulative probability threshold P.
- The model selects a smallest set of token whose total probability ≥ P.

) Input = "Our new smartwatch is packed with features including.."
Model's raw token probabilities (Top 6) are:

Token	Prob
Health	0.35
Fitness	0.22
Sleep	0.18
Notification	0.12
GPS	0.08
Battery	0.05

let P = 0.5 , ⇒ 0.57 (Cumulative score of Health & Fitness)
let P = 0.8 , ⇒ 0.87 (Cum. score of Health, Fitness, Sleep, Notification)

28-11-25 ⇒ Advanced Prompt Engineering :

• It is a process of designing smart, multi-steped & structured prompts that helps LLMs to solve complex problems with higher accuracy & better reasoning.

• Prompt Engineering becomes advanced when it goes beyond simple instructions & uses strategic, multi-step & structure techniques to perform LLM to perform complex tasks accurate tasks instead of giving commands, advanced prompt engineering focuses on how the model thinks & verifies the output.

• Advanced Prompt Engineering uses methods like -

i) Zero-shot prompting : It is the simplest type of prompt that provides a description of a task & some texts for the LLM to get started with.

ii) Few-shot prompting : Two or more examples are provided to demostrate desired behavior

iii) Instruction prompting : Providing a clear & explicit set of rules or commands. For eg- "Summarise the text in 3 points within 50/100 words."

iv) Role based prompting : It is a technique in prompt engineering that involves assigning a specific role to the AI model, this can help the model to generate more relevant & informative output.

v) Chain of Thought prompting (CoT) : It is a technique for improving the reasoning capabilities of LLMs by generating intermediate reasoning steps, this helps the LLM to generate more accurate answers.

vi) Tree of Thought prompting (ToT) : It encourages to explore the multiple solution paths.

vii) Self Consistency prompting : Using the same prompt multiple times can lead to different results. To resolve this degree of randomness & improve the performance of generative models, Self Consistency concept was introduced. This method asks the generative model the same prompt multiple times & takes the majority result as final answer.

11-25

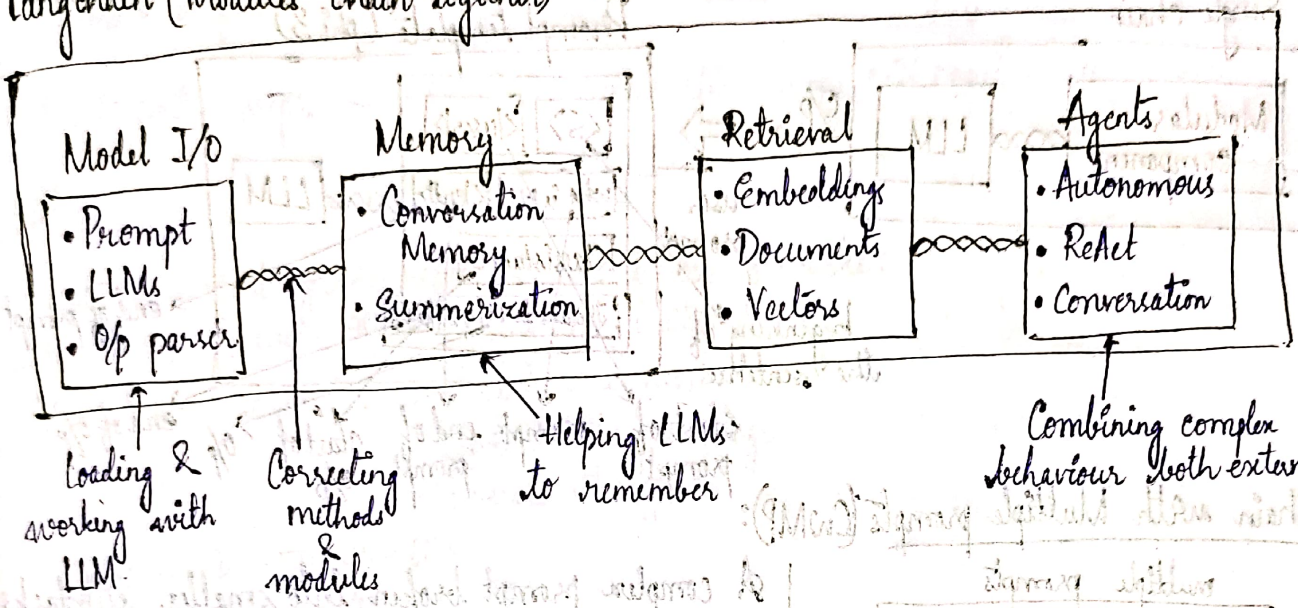
Chapter-01

Advanced Text generation Technique & Tools

This chapter explores the methods to improvise the quality of the generated text.

→ Framework

langchain (Modules chain together)



1.

⇒ Model I/O: Loading quantized models with langchain

i) It refers to how we can load, learn & manage input/output of language models.

ii) It deals with

a) Loading modes from local files or hubs.

b) Using transformers library to handle tokenizers & model inference.

c) Co-ordinating i/p text → tokens → Model → o/p tokens → Final text.

iii) Quantized models are machine learning or LLM models that has been compressed by reducing the precession of its numeric values (weights to make the model smaller, faster & cheaper to run.

Eg: Original weight (FP32) = 0.092345789

↓

Quantized into INT8

= 0.09

iv) Quantization is the process of reducing model with precision from 32-bit floats or FP32 to lower bit format i.e, INT8, INT4, INT2.

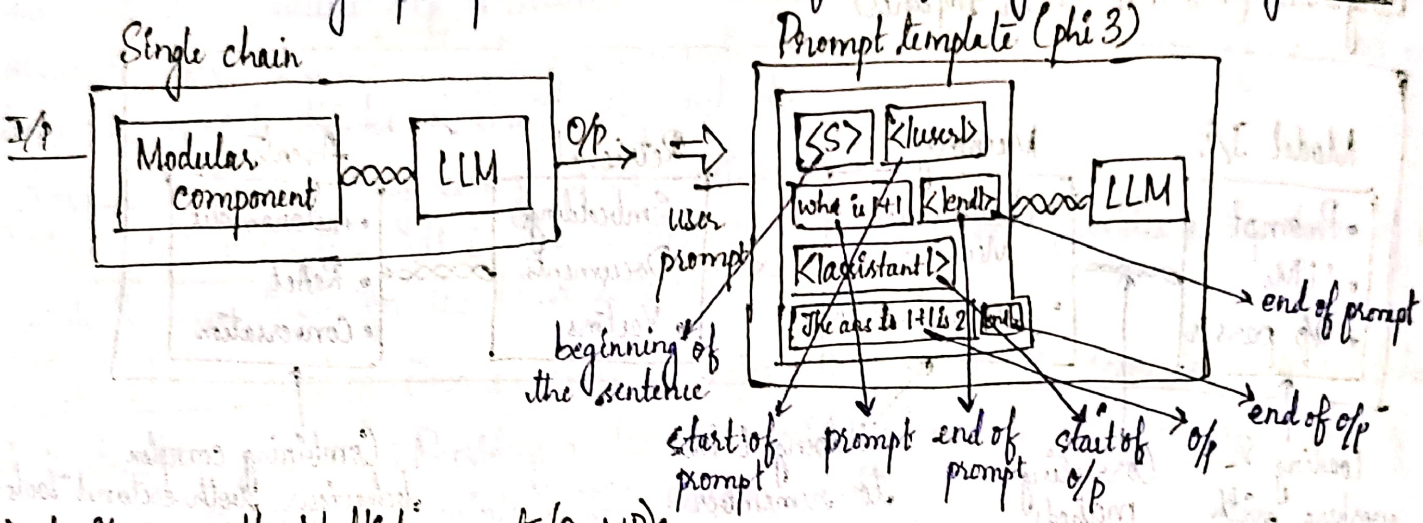
01-12-25

2⇒ Chain: Extending the capabilities of LLM:

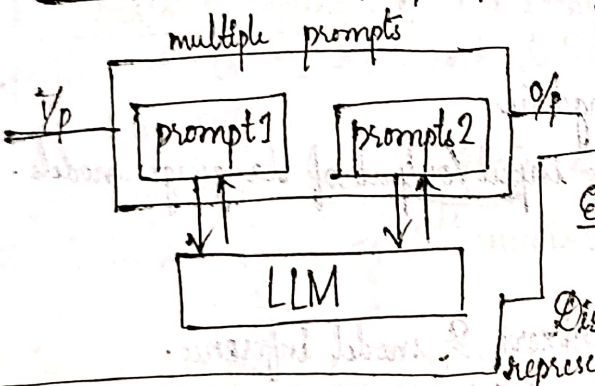
Chain → most imp/core functionality of langchain

main method

- LangChain is named after its main method i.e. Chain.
- Chains are multi-step sequences that connect several operations together to complete a task using an LLM. Each step can be a prompt, a model call, a function ~~call~~ or tool, a data preprocessing step, etc. A chain allow LLM to solve complex task that cannot be solved in a single prompt. The most basic form of langchain is a Single chain.

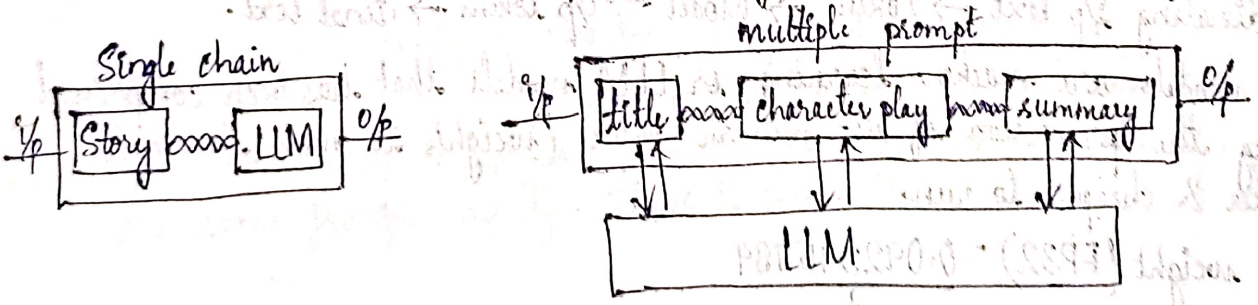


⇒ A Chain with Multiple prompts (CwMP):



A complex prompt broken into smaller subtasks that can run sequentially. This require multiple calls to LLM.

Eg: Generate a story with the complex details like title description of character, under summary. Dissect this prompt into manageable smaller tasks & represent its general structure through multiple prompts



04-12-25

⇒ Differentiate between COT, Chain with Multiple prompt, Prompt Chaining:

	COT	CwMP	PC
Def:	• It is a method where prompt produces intermediate reasoning steps to reach final answer.	• A single task is completed using multiple prompts, sent in a structured sequence, often within one model call or prompt template	• A full task is broken into separate steps & the o/p of one model call becomes the i/p of the next.
Purpose:	• Asks model to think step by step	• Multiple commands stored in 1 template	• Split complex tasks into steps
No. of prompts:	• 1, One prompt	• Multiple prompts	• Multiple prompts
No. of model calls:	• 1, One call	• 1, one call	• Multiple calls

CoT

Structure:
- T_1: prompt (think step-by-step)
  ↓ one call
  LLM generate CoT + Final o/p

Pros:
- It is simple & fast.

Cons:
- Harder to control steps.

CwMP

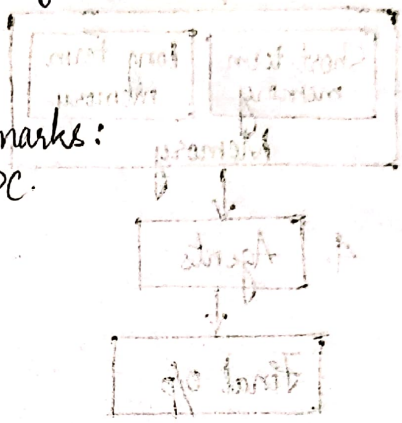
- T_1: prompt 1 + prompt 2 (concatenated)
  ↓ combined call
  LLM o/p

- Easy to iterate & interact.
- Depends on conversation memory.

PC

- T_1: prompt 1 → LLM → o/p 1
  T_2: prompt 2 (using o/p 1) → LLM → o/p 2
  T_3: prompt 3 (using o/p 2) → LLM → Final o/p

- Highly controlled & good for automation
- It requires design effort.



06-12-25

Q) Generate a student grade report for the following marks:
Math - 45/50, Science - 40/50, using CoT, CwMP & PC.

A: Student grade report :
1. Calculate total mark
2. Findout the percentage
3. Generate the grade

Total mark = 50 + 50 = 100
Marks obtained = 45 + 40 = 85
Percentage = $\frac{85}{100} \times 100 = 85\%$ → 'B' grade

CoT

prompt → Calculate the tot-mark %& grade for math-45/50, sci-40/50
Think step by step & show the reasoning steps.
(single call ↓)

LLM o/p: Step1 - Total mark = 85
Step2 - Maximum mark = 100
Step3 - Percentage = 85%
Step4 - Grade = B

Final answer = Total = 85, Percentage = 85
Grade = B

CwMP

<task> Generate grade report for marks math - 45/50, sci - 40/50 </task> <marks>

<instruction>
1. Compute Tot mark
2. Find out the percentage
3. Assign grade
</instruction>

LLM o/p: Total mark = 85
Percentage = 85%
Grade = B

(single call ↓)

T_1

1.	2.	3.
P_1	P_2	P_3

PC

Call 1: Extract raw data
prompt 1: Extract the marks of math 45/50 & sci 40/50

LLM o/p 1: math - 45
sci - 40

(i/p ↓)

Call 2: Calculate tot-marks & %ge
prompt 2: math - 45
sci - 40
Calculate tot-mark & %ge

LLM o/p 2: Total marks = 85
Percentage = 85%

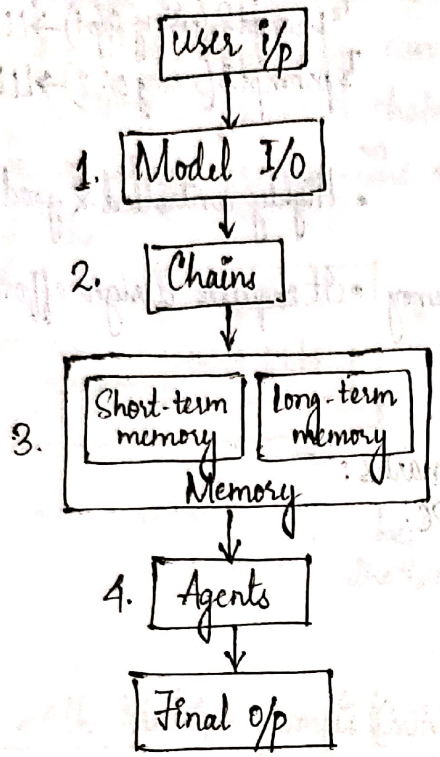
(i/p ↓)

Call 3: Assigning grade
prompt 3: Assigning grade for tot mark

LLM o/p 3: Percentage = 85%
Grade = B

Final o/p: Total mark = 85
Percentage = 85%
Grade = B

06-12-25



3.

⇒ Memory: Memory stores & retrieves information, so that LLM can maintain content across steps, sessions or tasks in langchain. There are two types of memory:

i) Short-term memory: It helps the LLM to remember past earlier discussions (past few min/hours). It stores conversation history & recent context.

ii) Long-term memory: It helps to retrieve the facts & the knowledge. It doesn't forget the conversation after the chat ends. It uses vector database for storing of information (RAG).

06-12-25

(continuation to previous content - Long term mem)

There are 2 common methods for helping LLMs to remember conversation.

i) Conversation Buffer: One of the most intuitive forms of giving LLMs memory is simply reminding them exactly what has happened in the past. In LangChain this form of memory is called a Conversation-Buffer-Memory.

a) Windowed Conversation Buffer: It is the method used in ChatBots or LLM appl^n to store only the recent part of conversations instead of entire chat history. It is mainly used to reduce memory usage, control context length & improve performance.

ii) Conversation Summary: It is a technique used to manage long chats by compressing earlier parts of the conversation into a short meaningful summary. This helps the model stay consistent without exceeding token limits.

→ Why it is needed: The Conversation Summary help with:

i) Token efficiency: Summaries are far shorter than full history.
ii) Maintaining coherence: The model remember user goals & previous context.
iii) Scaling to long conversations: It enables hrs, days of conversation history.
iv) Supporting agent based system: Agents rely heavily on memory to plan multi-steped tasks.

4.
⇒ Agents: An Agent is a LLM powered decision maker that chooses what to do next based on the user's i/p & the tools available.
• An Agent in LangChain is LLM + tools + reasoning loops that lets the model choose the next action dynamically.

→ Why agents are needed?:
Agents solve the problem by:
i) Understanding the user's query
ii) Deciding which tool to use
iii) Calling that tool
iv) Looking at the result
v) Deciding the next step
vi) Repeating until the task is completed
This makes the agent dynamic, flexible & intelligent.

11-12-25

→ Components of Agent in LangChain:
i) LLM (Brain): The language model that thinks & decide.
ii) Tools (Actions): Calculated tool, Websearch tool, database tool, python tools.
iii) Agent Executor: The controller that loops like think → act → observe → think → ...

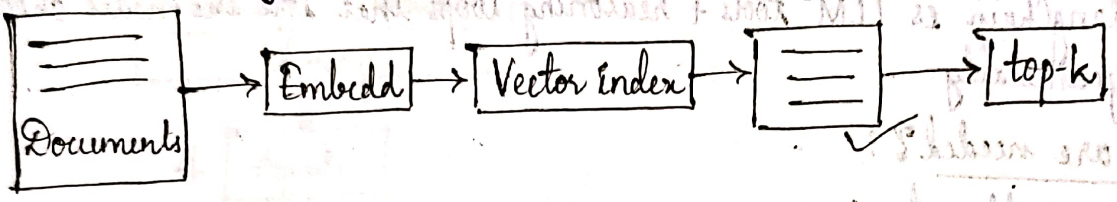
⇒ ReAct: It is combination of two words - Reasoning & Action
It is a process or framework or methodology that defines how a LLM should behave when solving problems, that requires step by step reasoning, calling external tools or actions, reading (observe) results then decide the next action. Without LLM, ReAct cannot work & without ReAct, LLM cannot dynamically act with tools.
Togetherly LLM & ReAct create an autonomous problem solving system & it tells the LLM how to think & act & the LLM provides the intelligence to follow the ReAct workflow.

Chapter-08

Semantic search & Retrieved Augmented Generation (RAG)

⇒Semantic Search :

- It is a way of retrieving information based on meaning rather than just keyword matching.
- It converts queries & documents into vector embeddings that capture conceptual similarity
- The system then compares these vectors & returns results that are semantically related, even if they use different bodies wordings.



⇒Difference between Keyword Search & Semantic Search :

Features:	Keyword Search	Semantic Search
Purpose	• Looks for exact or stem words or phrases matches in the index.	• It converts text into dense vectors & compares the meaning using similarity score.
Tech. used	• BoW, TF-IdF, BM25, Inverted Index, Keyword Lookup.	• Embeddings, Vector similarity, Neural Models
Understand synonyms	• It doesn't understand synonym of the word.	• It understands synonym of the word.
Strength	• Simple & fast to compute; transparent & works well with the rare terms.	• It capture synonym, paraphrases & context; handles fuzzing intent
Weakness	• It misses related concepts; sensitive to spelling or word choice	• It requires a good embedding model & more storage.
Search space	• Lexical or words.	• Semantic or meaning
O/p quality	• Depends on exact word overlap.	• Depends on conceptual similarity.

⇒ Semantic Search : It is of two types :-

(i) Dense retrieval : It is a retreival method that uses dense vector embedding to find documents that are semantically similar to a user's query.

→ Why it is required? - Because it understands meaning but not just exact words. It handles synonyms. Works well for long, natural language queries & also captures contents & relationships between terms.

→ Limitation -

- Answer must exist in corpus(db).
- (DR cannot retrieve correct answers if the inf^n isn't present in index-text).
- Weak at exact matches
(DR may miss precise keywords, phrases, names or codes).

- Domain Sensitivity
(Performance drops when embeddings are applied to domains different from their training data).
- Chunking Dependent
(Retrieval quality heavily depends on how documents are split into chunks).

13-12-25

Eg: Consider a small document & process using Dense retrieval concept
Doc - Semantic search helps computers understand meaning. Dense retrieval uses vector embeddings. Text is broken into chunks for efficient search

Step1: Break the document into chunks.
Semantic search system always split long text into small pieces called chunks, because embedding models works best with short & focused text. The chunks are -

chunk1
| Semantic search helps ... |

chunk2
| Dense retrieval uses ... |

chunk3
| Text is broken .. |

Step2: Convert each chunks into embedding vectors, i.e -

C1: | 0.21 | 0.55 | 0.13 | .. |

C2: | 0.89 | 0.1 | 0.66 | .. |

C3: | 0.32 | 0.69 | 0.05 | ... |

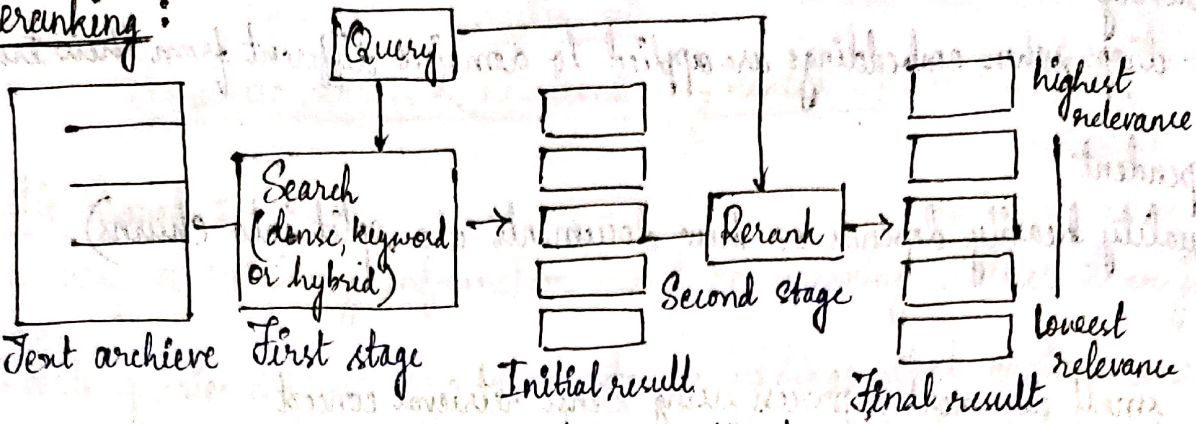
Step3: Process a query
Let Q: What uses embeddings?

Step4: Convert a query into an embedding form

Q: | 0.36 | 0.42 | 0.51 | .. |

Step5: Dense retrieval by using/finding nearest vectors.
(Q, C1) - 0.21
(Q, C2) - 0.91 ✓
(Q, C3) - 0.18

(ii) Reranking :



- In Semantic Search system, Reranking is the two-stage refinement step that takes the initial set of search results & reorders them based on differ deeper semantic relevance.
- Reranking is needed because the first passed dense retrieval is fast but only approximates relevance. However, it can miss subtle intent, handle synonyms imperfectly or be fooled by high frequency words.
- A second stage is more expensive model that looks at the query & each candidate together, capture contextual meaning, the context & the domain specific meaning, & reorder the results so that the top few are truely the most usefull.

15-12-25

⇒ Retrieval Evaluation Metrics :

i) It measures how many of top k results return by a system are actually relevant.

$$\text{Precision at } K = \frac{\text{No. of relevant doc in top } k}{k}$$

ii) Average Precision (AP) - Avg of Precision value is computed at each rank where a relevant document appears.

$$AP = \frac{1}{R} \sum_{i=1}^{N} P(i) \cdot rel(i)$$

iii) Mean AP (MAP) - It is the arithmetic mean of AP scores across a set of queries. It gives a single query independent measure of a ranking system's overall performance.

$$MAP = \frac{1}{Q} \sum_{j=1}^{Q} AP_j$$

iv) Normalized Discounted Cumulative Gain (nDCG) - It measures how well a rank list of items matches an ideal ordering, taking into account both relevance & position.

$$nDCG = \frac{DCG}{IDCG} \quad \text{where} \quad DCG = \sum_{i=1}^{n} \frac{relevance_i}{\log_2(i+1)}, \quad IDCG = \sum_{i=1}^{n} \frac{Grade_i}{\log_2(i+1)}$$

here, sort doc, relevance = 1
relevance = 0

sort doc, Grade is either binary (i.e 1/0) or can consider 3pt/5pt rating scale.

8) An user issues a query & the retrieval system returns 5 documents, ordering from top to bottom by mentioning relevant or not relevant.

Rank	Doc	Relevant?	Relevant Grade	
1	A	Rel	1	1
2	B	Not Rel	0	1
3	C	Rel	1	1
4	D	Not Rel	0	0
5	E	Rel	1	0
				(sorted)

Step 1 : Precision at k=3 (tot) = $\frac{2}{3}$

Step 2 : $AP = \frac{1}{3}\left(1(1/1) + 2(1/3) + 3(1/5)\right) = \frac{1}{3}\left(1 + \frac{2}{3} + \frac{3}{5}\right) = \frac{84}{45} = \underline{0.75}$.

Step 3 : MAP = 0.75

Step 4 : $DCG = \sum_{i=1}^{n} \frac{relevance_i}{\log_2(i+1)} = \frac{1}{\log_2(1+1)} + \frac{0}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} + \frac{0}{\log_2(4+1)} + \frac{1}{\log_2(5+1)}$

$= 1 + 0 + 0.5 + 0 + 0.38 = \underline{1.88}$.

$IDCG = \sum_{i=1}^{n} \frac{Grade_i}{\log_2(i+1)} = \frac{1}{\log_2(1+1)} + \frac{1}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} + \frac{0}{\log_2(4+1)} + \frac{0}{\log_2(5+1)}$.

$= 1 + 0.63 + 0.5 + 0 + 0 = \underline{2.13}$. $nDCG = \frac{1.88}{2.13} = \underline{0.88}$.

9) An IR system produces the following ranking in answer to queries q_1 & q_2. The underscored documents are the ones relevant to the user.

(underlined doc are relevant doc)

RG:q_1	q_2	R	q_1	q_2
1	1	1	A	F
0	1	2	L	G
1	0	3	G	D
0	1	4	F	E
0	1	5	D	L
1	0	6	E	I
1	0	7	B	H
1	0	8	H	C
0	1	9	I	B
0	0	10	C	A

i) Precision at k

$P_{q_1} = \frac{2}{5}$ $P_{q_2} = \frac{3}{4}$

ii) $AP_1 = \frac{1}{5}\left(1(1/1) + 2(1/3) + 3(1/6) + 4(1/7) + 5(1/8)\right) = \underline{0.65}$.

$AP_2 = \frac{1}{4}\left(1(1/1) + 2(1/2) + 3(1/4) + 4(1/9)\right) = \underline{0.79}$

iii) $MAP = \frac{0.65 + 0.79}{2} = \underline{0.72}$

iv) $DCG_1 = \frac{1}{\log_2(1+1)} + \frac{0}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} + 0 + 0 + \frac{1}{\log_2(6+1)} + \frac{1}{\log_2(7+1)} + \frac{1}{\log_2(8+1)} + 0 + 0$

18-12-25

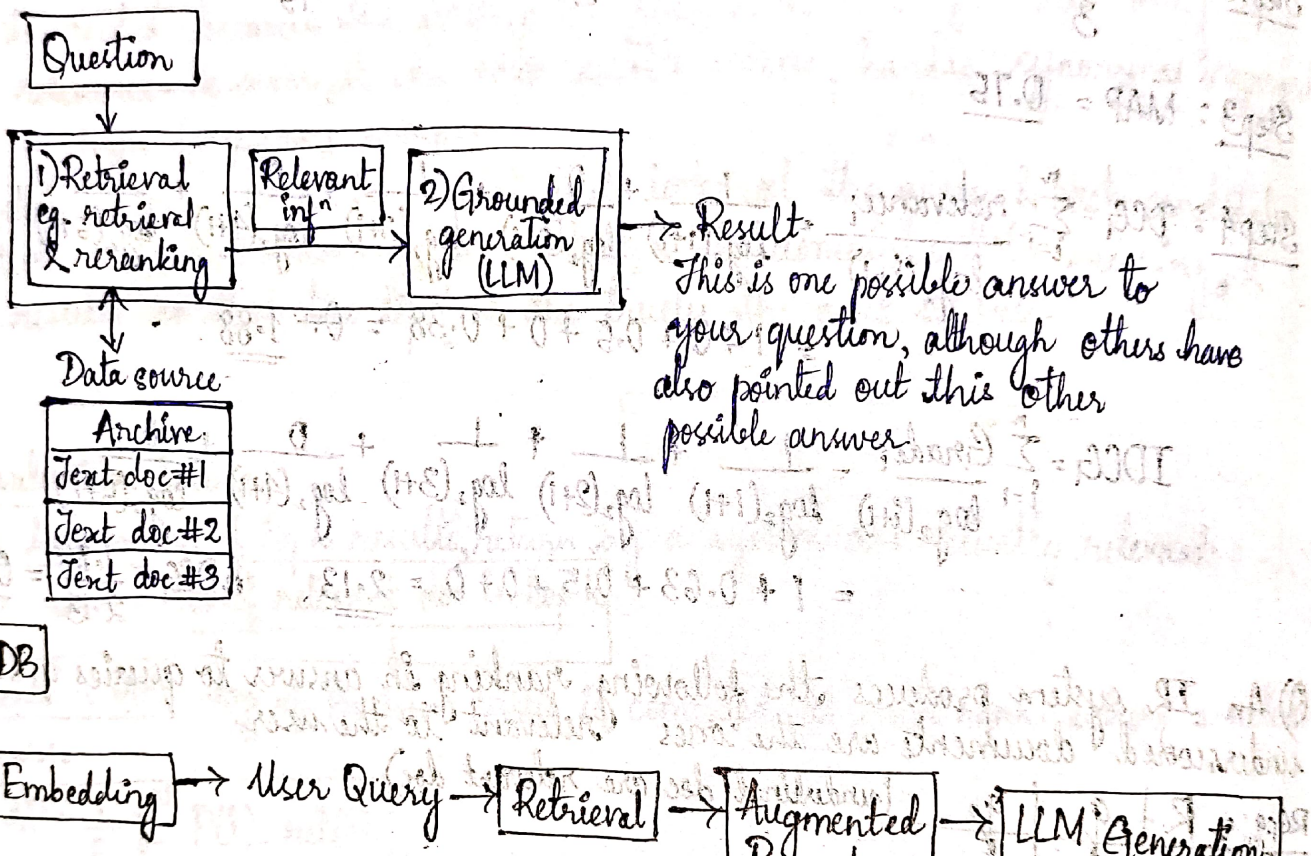
⇒ Retrieval Augmented Generation (RAG):

It is a framework that enhances large language models by retrieving semantically relevant document chunks from an external knowledge store & injecting them into a prompt, allowing the model to generate fact-grounded & upto date responses without retraining.

19-12-25

Q) Given a vocabulary size of 40,000 & an embedding dimension of 768, calculate the no. of parameters in the embedding layer. Multiplying it we will get no. of parameters.

⇒ RAG:



⇒ Document Processing:

Step 1: Chunking
Step 2: Embedding Vectors
Step 3: User Query
Step 4: Similarity Computation
Step 5: Retrieval Result
Step 6: Content integration (or Prompt construction)
Step 7: Generate Answer

→ Advanced RAG Techniques:

- Before retrieval
- During retrieval
- After retrieval
- During generation

User Query
↓
Query Rewrite / Expansion
↓
Hybrid Retrieval (Dense + Sparse)
↓
Reranking
↓
Content Filtering & Compression
↓
Augmented Prompt
↓
LLM Generation
↓
Self Verification / Check-up

Query Rewriting: LLM rewrites the query into a clearer, retrieval-friendly form

→ Multi-query retrieval: Extension of query retrieval. The next improvement we can introduce is to extend the query rewriting to be able to search multiple queries if more than one is needed to answer a specific question.

Generate multiple variations of the query & retrieve documents for all.

Q_1: What is dense retrieval?
Q_2: How does dense retrieval work?
Q_3: Dense retrieval

21-12-25

→ Multi-hop RAG:-

Eg: User Question - "Who are the largest car manufacturers in 2023? Do they each make EVs or not?"

- To answer this, the system must first search for
  Who are the largest car manufacturers in 2023.
- Then after it gets this information (the result being Toyota, Volkswagen, & Hyundai), it should ask follow-up questions:

Step 2, Query 1 - "Toyota car makes EVs"
Query 2 - Volkswagen

→ Query routing:-

It simply does redirection
It is process of classifying a user query & directing it to the most suitable retrieval strategy or data source.

→ Agentic RAG:

A language model acts as an agent, that means LLM plans to what it will do, then decide what to process, ...

→ Basic RAG (static) –

Query → Retrieve → Augment → Generate

→ Agentic RAG (dynamic) –

Query

↓

Plan (LLM decides next step)

↓

Retrieve / Call Tool → Retrieve information

↓

Check / Reflect

↓

Retrieve again (if needed)

↓

Final Answer

→ RAG Evaluation :

This is to check how far the answer is appropriate to query.

RAG System

└ Retrieval Evaluation (Precision at k, AP, MAP, nDCG).
↳ Basic evaluation strategy

└ Generative Evaluation ()

↳ It evaluates results along four axes : Fluency : whether
Perceived utility
Citation recall
Citation precision