LLM (Large Language Model)

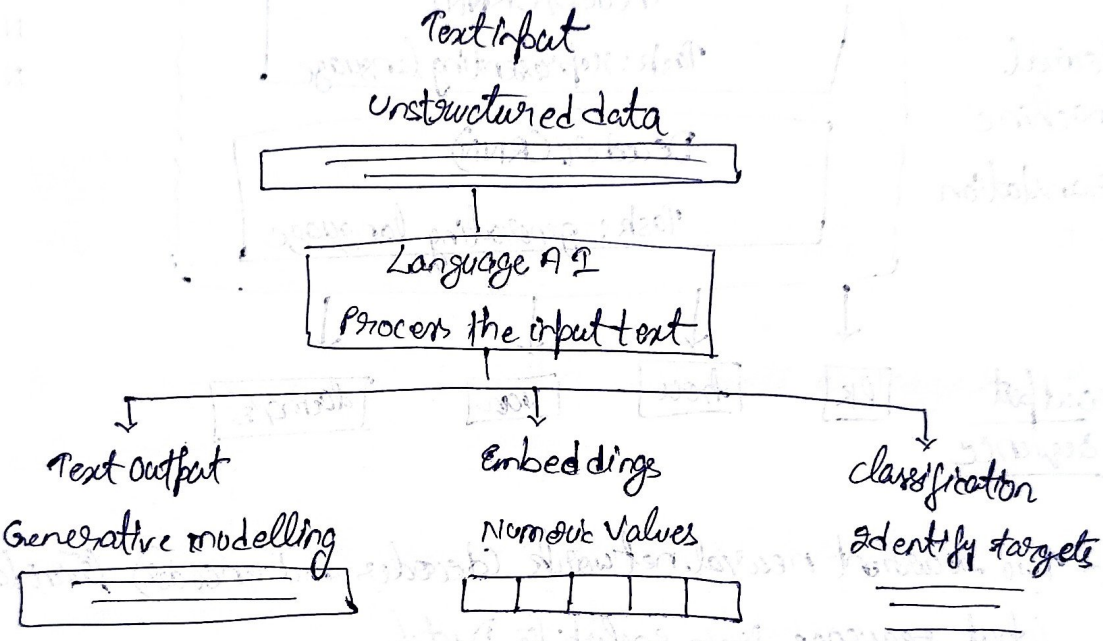
- Deep Learning is a subset of AI.
- Language AI is used to process and generate human language.

* After 2018, the AI models uses transformers (convert one language to another).

* Transformer can be Decoder, encoder or both encoder and decoder.

LLM – The deep learning based model used transformer to train on the natural language like text or image.

- The language AI is the subfield of AI used to understand process and generate the human language.



Bag of words – A method for representing unstructured text.

Steps 1> Tokenization – Splitting the sentences into individual words or subwords is known as tokenization ~~(token)~~. (tokens)

Input
This is a cute dog
↓ split i/p by whitespace
"that" "is" "a" "cute" "dog"
↓ create vocabulary
[that] [is] [a] [cute] [dog] [my] [cat]

Input
my cat is cute
↓
"my" "cat" "is" "cute"

numbers. This is why the tokenization comes into play.

Steps behind the tokenization -

1) Assign each token a unique number.
2) These integers are then vectorized and feed into the language model.
3) The language model then process this vectorized input and outputs the integer number again, then the integers subsequently converted back into text. That is the generated output.

Imp

There are 3 types of tokenization -

i) Byte Pair encoding - Greedy merging of frequently adjacent character pairs to form subwords. Vocabulary is built by iteratively combining the most common pairs.

Eg: GPT, OpenNMT

ii) Word Piece - Selects subwords to maximize the likelihood of the tokenized sequence under a language model. often uses prefix indicators like for suffixes.

Eg: BERT, DistilBERT

iii) Unigram Language Model - Probabilistic model that assigns likelihoods to subword units and selects the most probable segmentation. Allows multiple segmentations per word.

Eg: XLNet, T5, Sentence Piece.

Eg: Byte pair coading example: aa abc abc.

Sol: Initial Corpus: ["aa", "abc", "abc"]

Step 1: Preprocessing

"aa": ["a", "a", "_"]

"abc": ["a", "b", "c", "_"]

"abc": ["a", "b", "c", "_"]

Step 2: Iterative merging

1. Iteration 1:

- Most frequent pair: ("a", "b")
- Merge into: "ab"
- Corpus updates:
  - ["a", "a", "_"]
  - ["ab", "c", "_"]
  - ["ab", "c", "_"]
- Vocabulary: ["a", "b", "c", "_", "ab"]

2. Iteration 2

("ab", "c")

- Merge into: "abc"
- Corpus update:
  - ["a", "a", "_"]
  - ["abc", "_"]
  - ["abc", "_"]
- Vocabulary: ["a", "b", "c", "_", "ab", "abc"]

3. Iteration 3

("a", "a")

- Merge into: "aa"
- Corpus update:
  - ["aa", "_"]
  - ["abc", "_"]
  - ["abc", "_"]
- Vocabulary: ["a", "b", "c", "_", "ab", "abc", "aa"]

Final Tokenized Corpus

- ["aa", "_"]
- ["abc", "_"]
- ["abc", "_"]

Final vocabulary: ["a", "b", "c", "_", "ab", "abc", "aa"]

- The decoder block in transformer ~~consists of~~ consists of
  i) Masked self-attention ii) Encoder attention iii) Encoder neural network.

9/9

- BERT is an encoder only model (Introduced in 2018).
- GPT 1 is an decoder only model. Trained with 7000 books and has 117 million parameters. (Proposed in 2018).

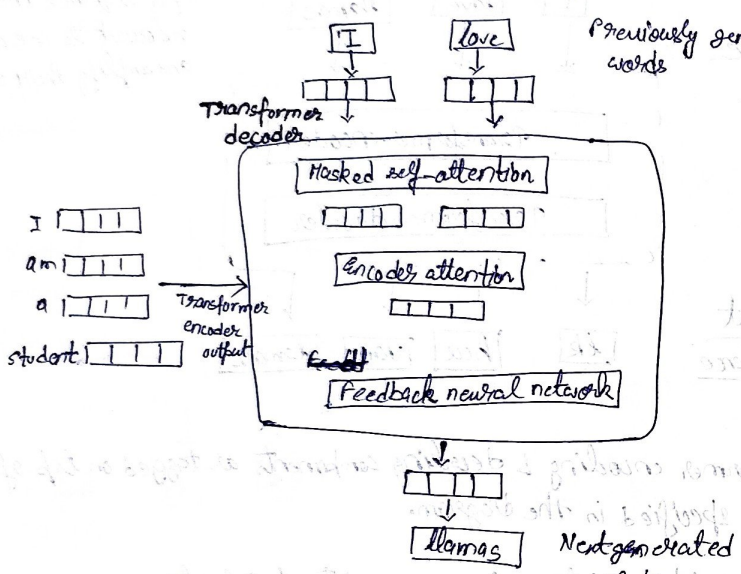


Fig 1.9 → The decoder has an additional attention layer that attends to the o/p of the encoder.

BERT - (Bidirectional encoder Representations from transformer)

- It's an encoder only architecture, ~~they~~ that are focus on representing language.

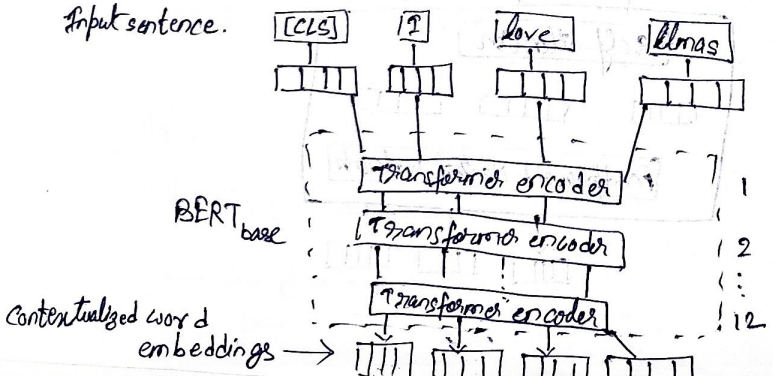


Fig 1.21 The architecture of bert with 12 encoders.

Q> How the LLM generate the tokens of text?

Sol:

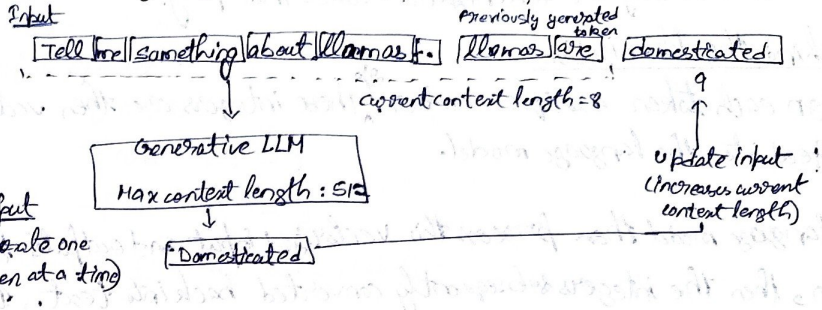


Fig 1.27 → The context length is the maximum context an LLM can handle

---

9/9/25

Chapter - 2 : Tokens and Embeddings

Tokenization and embeddings -

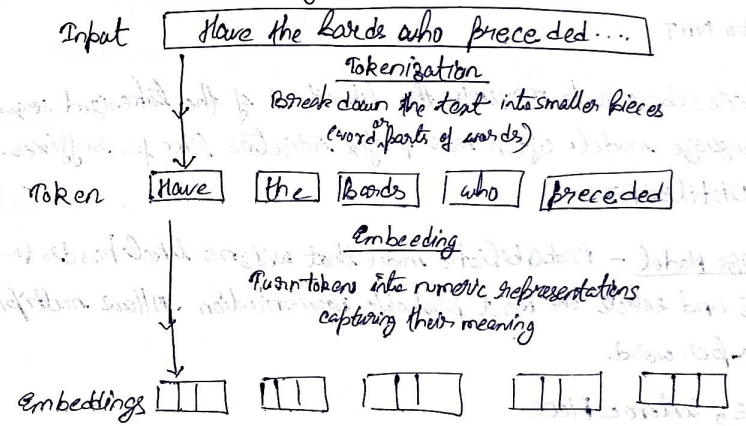


Fig 2.1 → Language models deals with text in small chunks called tokens. For the language model to compute language, it needs to turn tokens into numeric representation called embeddings.

Tokenization - is the process of splitting of sentence into small parts or sequence of tokens which can be either words, subwords or characters.

Q> Why tokenization is required?

Suppose, we want to pretrain our model and we have 10 GB of language data available.

However, the data is in text form but the language model process the data in numbers. Therefore, we need to convert the text into the

Input

My cat is cute

↓

[my] [cat] [is] [cute]

↓

that	is	a	cute	dog	my	cat
0	1	0	1	0	1	1

Bag of words — count individual words ← vector representation

Convert the text from english to dutch language -

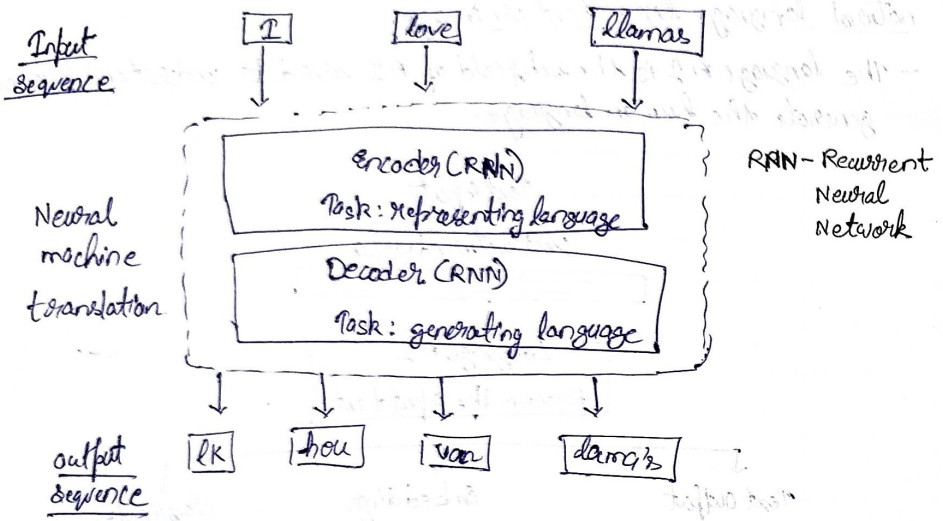


Fig - Two recurrent neural network (decoder and encoder) translating an input sequence from english to Dutch.

Embedding - process of converting tokens into vector forms which try to capture its meaning.

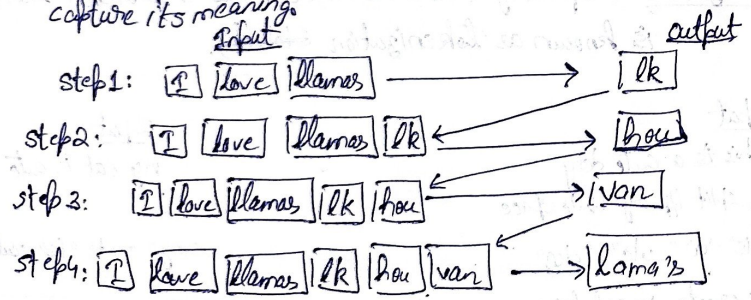


Fig 1.12 → Each previous o/p token is used as i/p to generate next token

Embedding - allows us to measure the semantic similarities b/w words.

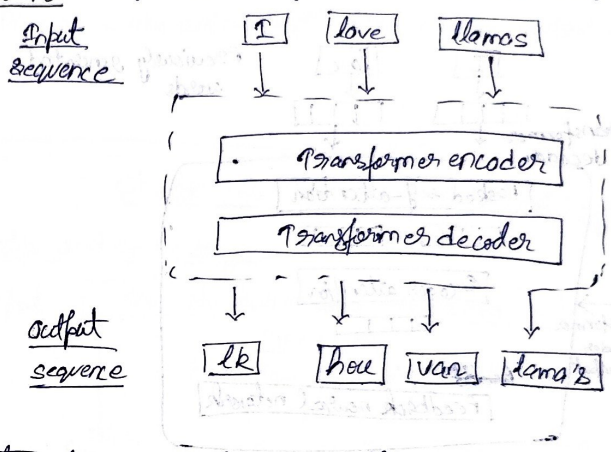
- Using various metrics we can judge how close one word to another

Types of embedding

i) Document embedding ii) Sentance embedding

iii) word embedding iv) Token embedding.

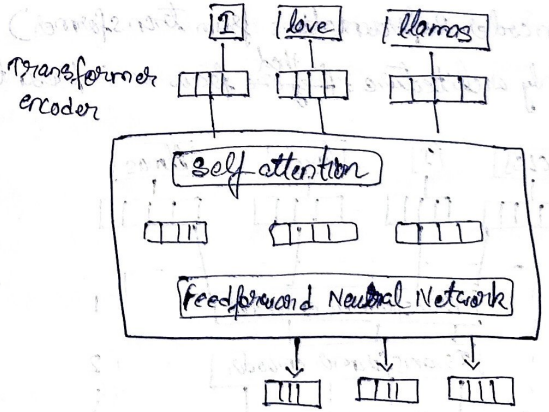
Attention - "Attention is all you need". It allows a model to focus on parts of the input sequence that are relevant to one another and amplify their signals.



In the transformer encoding & decoding components are tagged on top of each other as specified in the diagram.

- The encoder block in transformer consists of 2 parts -

i) Self attention ii) Feed forward network



Word Peice - The word peice tokenization was developed by google that the primary goal is to handle the unseen ~~words~~ words more effectively than traditional word level tokenization.

Suppose the vocabulary is given as [UNK, a, aa, abc, ##b, ##c]

Corpus ["aa", "abc", "abc"]

case 1: abc and ab are in vocabulary

- "aa":
  - Longest prefix match : "aa" ∈ vocab
  - Tokenized : ["aa"]
- "abc":
  - Longest prefix match : "abc" ∈ vocab
  - Tokenized : ["abc"]
- Final tokenized corpus : [["aa"], ["abc"], ["abc"]]

case 2: abc and ab are not in vocabulary

step by step tokenization of "abc":

1. Start at position 0 : Try longest match from vocab
   - Try "abc" → not in vocab
   - Try "ab" → not in vocab
   - Try "a" → match
   - Emit : "a"
2. Position 1 : Remaining string "bc"
   - Try "##bc" → not in vocab
   - Try "##b" → match
   - Emit "##b"
3. Position 2 : Remaining string is "c".
   - Try "###c" → match
   - Emit : "##c"

Final Tokenization : ["a", "##b", "##c"]

Note : Word Piece doesn't backtrack or look ahead to match "##bc" unlike it's greedy in the longest match.

Sentence Peice - is language independent sub word tokenization method developed by google which is designed to train directly on raw text without requiring the pre-tokenization.

- unlike BPE and wordpeice, the sentence peice treats the raw string of unique code characters for tokenization.
- The sentence peice use unigram language model that assigns a probability 'Pi' to each subword 'ti' in the vocabulary - to find the total cost.

$$\text{cost}(t_i) = -\ln P_i$$

Eg: Suppose sentence is "abc". Use sentence peice algorithm to tokenize the "abc".

Token	Probability P	cost = $-\ln P_i$
a	0.20	1.61
b	0.20	1.61
c	0.20	1.61
ab	0.15	1.90
bc	0.15	1.90
abc	0.10	2.30

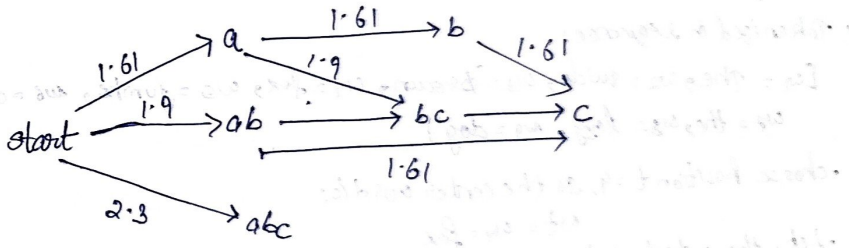


Fig 4: The cost lattice for "abc"

Segmentation : ["a", "b", "c"]
Cost : 1.61 + 1.61 + 1.61 = 4.83

Segmentation : ["abc"]
Cost : 2.30

Segmentation : ["ab", "c"]
Cost : 1.90 + 1.61 = 3.51

Segmentation : ["a", "bc"]
Cost : 1.61 + 1.90 = 3.51

Fig 3.10 - When generating text, its important to cache the computational results of previous tokens instead of repeating the same calculation over and over again

Transformer block - The transformer LLMs are composed of series of transformer blocks and each block process its input then passes the results of its processing to the next block.

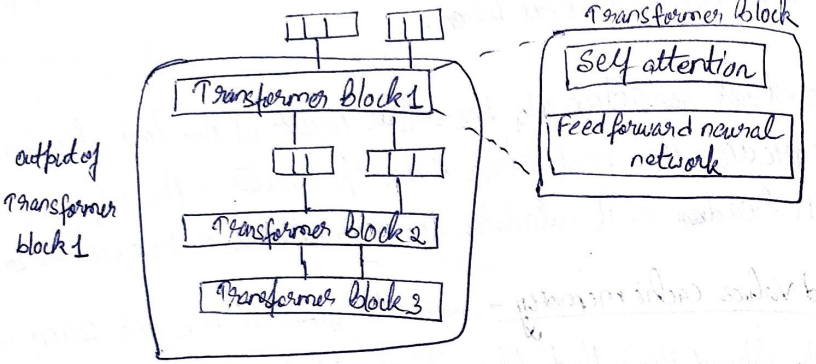


Fig 3.12 - A transformer block is made up of a self-attention layer and a feedforward neural network.

The shawshank - is a movie. When we search "The shawshank" it will automatically complete the whole movie name "The Shawshank Redemption"

19/9

Attention layer - Attention is a technique that used to process the input and generates the output according to the previous inputs.

- The self attention layer is mainly concerned with incorporating relevant information from other input tokens and positions.
- The feed forward layer is layer finds the majority of the models' processing capacity.
- The self attention mechanism is made of 2 major steps
  i) Relevance scoring
  ii) Feed forward & Combining information

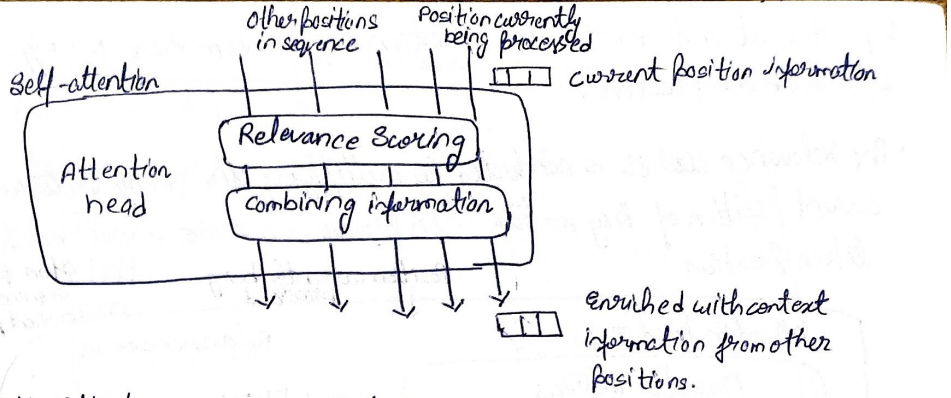


Fig 3.16 - Attention is made up of 2 major steps: relevance scoring for each position, then a step where we combine the information based on those scores

Attention head - contains all the parallel applications of attention into the self attention layer to give the transformer more extensive attention capability.

Q) How attention is calculated?

- The attention layer in generative LLM used for processing the attention for a single position.
- The inputs to the layer are
  i) Vector representation of the current position information.
  ii) Vector representation of previous tokens.
- The goal here is to produce a new representation of current position that incorporates relevant information from the previous tokens.
- In further the training process produces 3 major projection matrix to interrupt with the attention head for the calculations.
  i) Query projection matrix
  ii) A key projection matrix
  iii) A value projection matrix.

~~Relevance Query~~

Relevance scoring - In generated transformer every time it will generate only 1 token that means the processing will be always

Chapter 3 - Looking Inside of Large Language Models

Autoregressive model - In machine learning, the models that consume their early predictions to make later predictions means the models first generate the token is used to generate the second token are known as "auto regressive models".

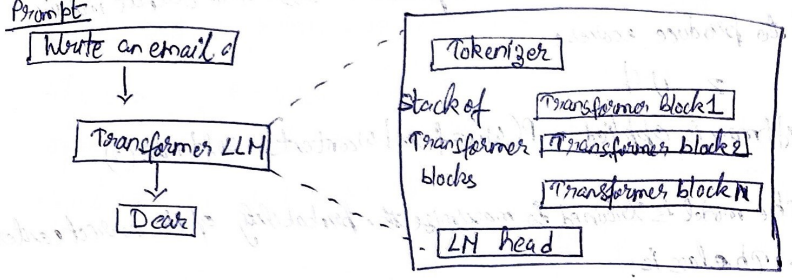


Fig 3.4 - A transformer LLM is made up of a tokenizer, a stack of transformer blocks, and a language modelling head.

Decoding strategy - The method of choosing a single token from the probability distribution means the probability score for each token in the vocabulary is called as "Decoding strategy".

- Choosing the highest scoring token every time is called as "Greedy Decoding".

Eg:

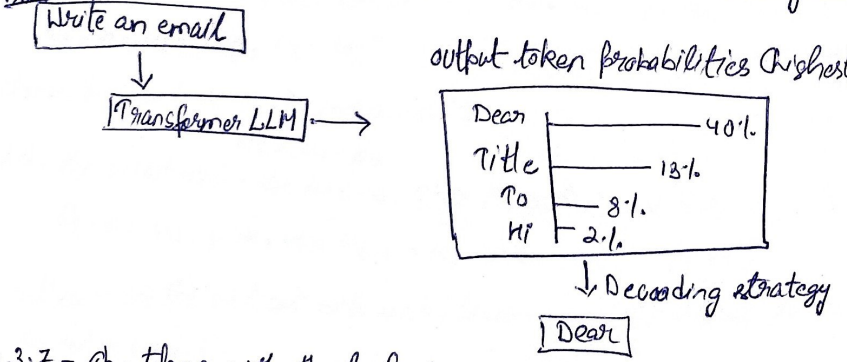


Fig 3.7 - The tokens with the highest probability after the model's forward pass. Our decoding strategy decides which of the tokens to output by sampling based on the probabilities.

Q) How the transformer LLM works?

Sol: i) First, the tokenizer in transformer LLM creates or converts the tokens which in further goes for token embedding means the vector representation.

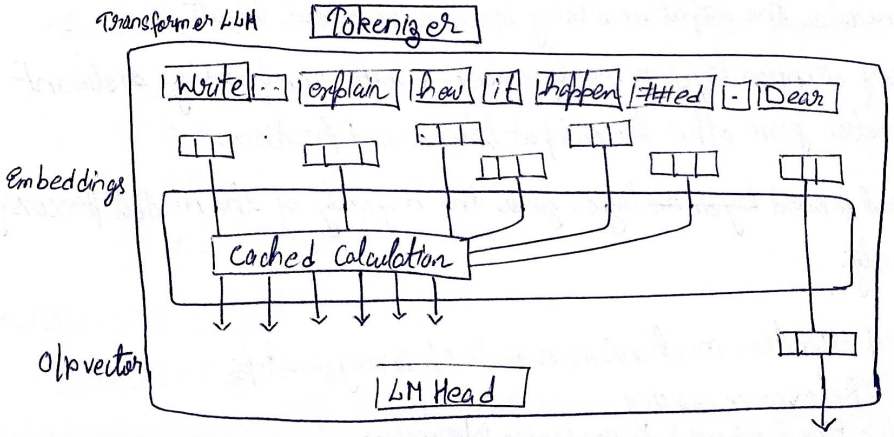
ii) Each generated token, the process flows once through each of the transformer blocks in the stack in order.

iii) That generated output goes to LM$_{head}$, which contains the probability distribution for the next token.

Prob - For text generation only the output result of the last stream is used to predict the next token that output vector is the only input into the LM head as it calculates the probabilities of the next token.

Keys and Values cache memory - When to generate the 2nd token we simply append the output token to the input and do the forward pass to the model. However, if we give the model the ability to cache the result of the previous calculation, we no longer need to repeat the calculation of the previous strings. This time the only needed calculation is for the last string. This process is called as "Key Value cache" which speeds of the generation process significantly.

- This is known as the "attention mechanism".



optimal path via Viterbi Decoding

The viterbi algorithm selects the path with minimum total cost.

In this case E["abc"] = 2.30

Best segmentation ["abc"], so total cost = 2.30

Embeeding - It will find the semantic similarity b/w 2 words using various matrices that can judge how close one word to another.

The embeeding technique has been used to convert the each token in input sequence into the numerical forms called as Input embedding.

- Each token is mapped to a vector using an embedding matrices.

$$x = \text{Embedding}(t)$$

t = token id

x = resulting vector

Word2Vec - It's the most embedding technique uses a sliding window to generate the semantic and syntactic relationship.

Eg: Consider the sentence: "The quick brown fox jumps over the lazy dog."

Let's define the context window and center word

- Tokenized in sequence:

[w_1 = the, w_2 = quick, w_3 = brown, w_4 = fox, w_5 = jumps, w_6 = over, w_7 = the, w_8 = lazy, w_9 = dog]

- choose position t = 4, so the center word is:

$w_t = w_4$ = fox

- Let, the context window size be C = 2. Then the context words are

$w_{t-2} = w_2$ = quick, $w_{t-1} = w_3$ = brown, w_{t+1} = jumps, $w_{t+2} = w_6$ = over.

CBOW objective - Use the context words quick, brown, jumps and over to predict the center word fox.

Embeeding and prediction steps -

1. One hot encoding - Each context word w_i is represented as a one-hot vector $x_i \in R^V$, where V is the vocabulary size.

2. Embedding lookup: Each one-hot vector is multiplied by the input embedding matrix $W \in R^{V \times N}$ to obtain its dense embedding

$$e_i = W^T x_i$$

2. Context aggregation: The embedding of four context words are averaged:

$$h = \frac{1}{4}(e_{quick} + e_{brown} + e_{jumps} + e_{over})$$

4. Prediction: The hidden vector $h \in R^N$ is passed through the output matrix $U \in R^{N \times V}$ to produce scores:

$$z = U^T h$$

Then softmax is applied: $P(w_t = fox \mid w_{context}) = softmax(z)$

5. Training: The model is trained to maximize the probability of the correct center word fox. The loss is:

$$L = -\log P(fox \mid quick, brown, jumps, over)$$

Result: After training, the rows of W become meaningful word embeddings. Word that appear in similar context (like fox and dog) will have similar embeddings.

1 position at a time. So, the attention mechanism here is only concerned with these one position.

- The relevance scores is conducted by multiplying the query vector with the current position of key matrix. This produces a score about each previous token position.

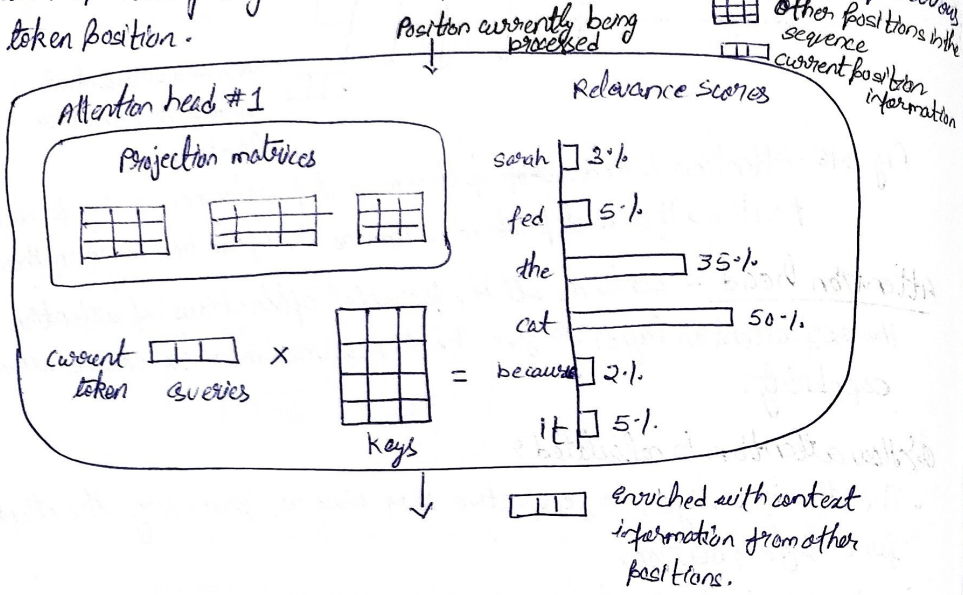


Fig 3.20 → Scoring the ~~relevance~~ relevance of previous tokens is accomplished by multiplying the query associated with the current position with the keys matrix.

- Passing these scores by a ~~subtract fun~~ softmax function normalizes these ~~sc~~ scores by ~~so~~ sum of ~~to~~ to 1.

Combining information - After performing the relevance scores the attention mechanism ~~multiply with~~ multiplied the scores with the value vectors for each token that means.

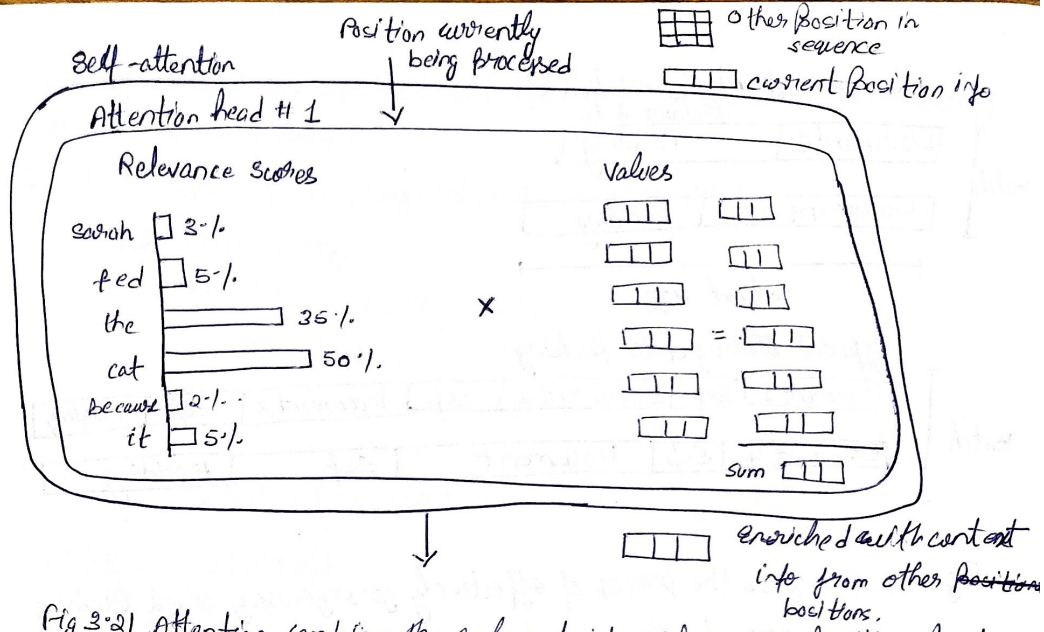


Fig 3.21 Attention combines the relevant info of previous positions by multiplying their relevance scores by their respective value vectors.

Local/sparse attention - To make more efficient attention, the local or sparse attention mechanism limits the context of previous tokens by only paying attention to a small no. of previous tokens / positions.

Positional embedding - also called as RoPE (Rotary Position Embedding).

- The RoPE is a key component in transformer LLM.

Q) Why positional embedding is required?

Sol: Let us take an example.

There are lot of documents are trained using language model, however the training set is much shorter than that context

- That means it would be inefficient to allocate the entire content.

  Ex: 4K ~~size~~ into a 10 word sequence.

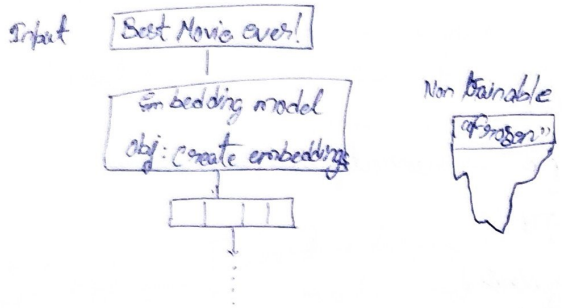
- So during model training the documents are packed together into each context in the training batch, that is known as packing.

our model make the prediction.

Classification task that leverage embeddings -

Instead of directly using the representation model for the classification, the embedding model uses 2 step approach for the text classification.

i) In the first step convert the text input into the embeddings using embedding model.



ii) These embeddings in the 2nd step serve as the input features to the classifier. The classifier is trainable and can perform classification.

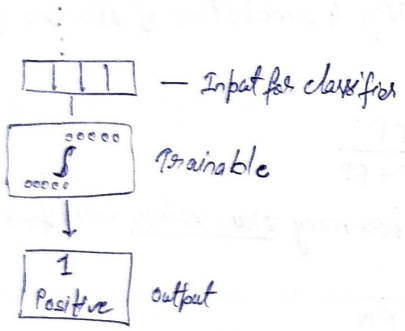


Fig 4:12

Benefit of this method is that -

We didn't need to fine tune our embedding model which can be costly.

10/10/25

Text classification using generative models -

The generative models ~~takes~~ takes as input some text and generate text in the form of sequence of tokens which is also known as sequence to sequence model. However, the task specific model takes the sequence of tokens as input but the model generates the numerical values instead of tokens.

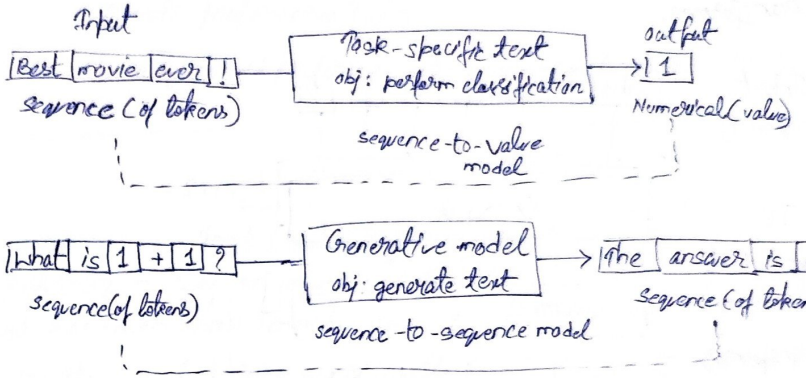


Fig 4:17 A task-specific model generates numerical values from sequences of tokens while a generative model generates sequences of tokens from sequence of tokens.

- These generative models are generally trained on a wide variety of task and usually don't perform out of box.

For eg: If we give a generative model, a movie review without any context then it has no idea what to do.

- This guiding process is done mainly through the instruction or known as "prompt" that we provided to the model.

- Iteratively improving the prompt to achieve the preferred result is called as prompt engineering.

- Prompt engineering allows prompts to be updated to improve the output generated by the model.

~~Text to Text~~

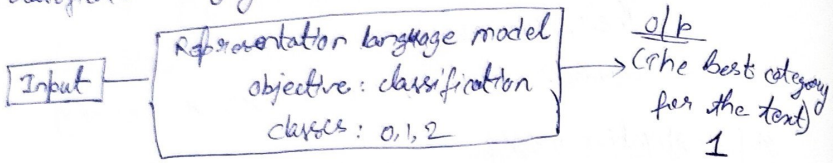
7/10/25

CH-4 : Text Classification

Text classification - classification of text is used to extract the meaningful insights like entities and detecting the language.

- The text classification in the field of language models can be represented using :
  i) Text classification with representation model
  ii) Text classification using generative model

Eg -



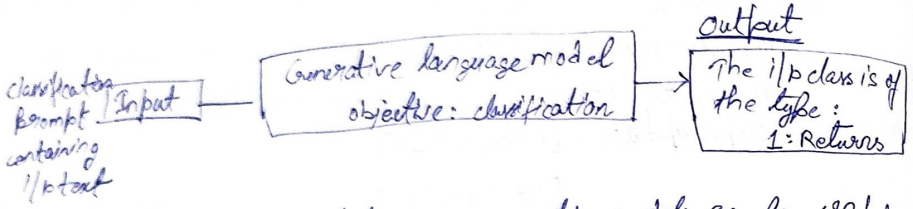


Fig 4:2. Although both representation and generative models can be used for classification, their approaches differ.

Text classification with representation models -

1) Task specific model
2) Embedding model

- The task specific model and embedding model are used for the text classification where the task specific uses tokenizer and token embedding to ~~generate~~ generate the word embedding.
- On the other hand the embedding model uses the supervised ML techniques for the text classification.

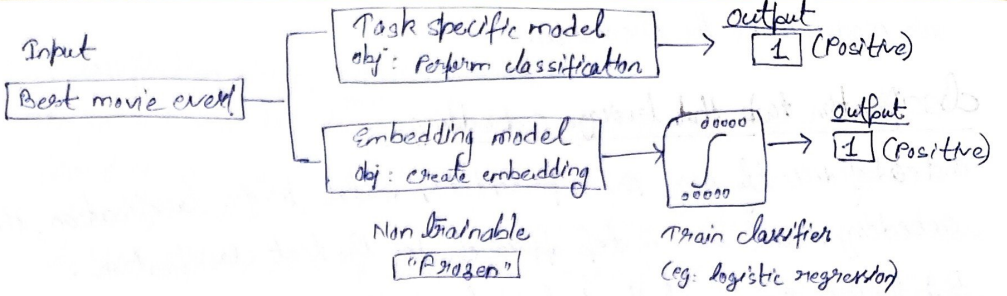
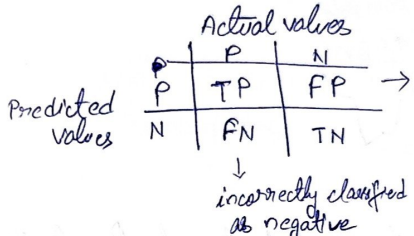


Fig 4:4. Perform classification directly with a task-specific model or indirectly with general purpose embeddings.

Confusion matrix



- The confusion matrix shows ~~what~~ whether the model correctly or incorrectly predicted result correspond to the actual values
- It represents by using 4 combinations of true and false values i.e TP, TN, FP, FN

$$\text{Precision} = \frac{TP}{TP + FP}$$

- It specifies how many true values identified actually.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- It defines out of all values which are actually predicted true.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

- Accuracy find how many are true out of all predictions.

$$\text{F1 score} = \frac{2(\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

- It combines both recall and precision scores which measures how effective

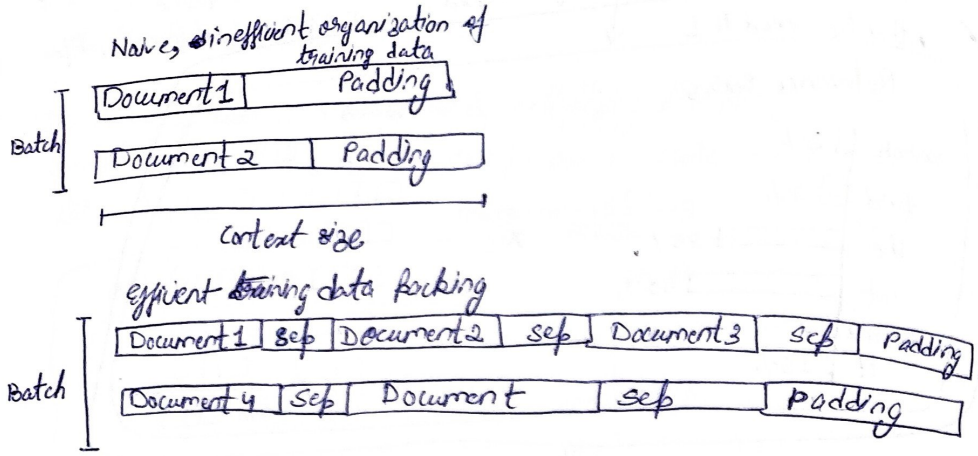


Fig 3.31 Packing is a the proces of effectively organizing short training documents into the context. It includes grouping multiple documents in a single context while minimizing the padding at the end of context.

Rotary Embeddings -

Rotary embeddings are a method to encode positional information in a way that captures absolute and relative token position information. It is based on the idea of rotary vectors in their embedding.

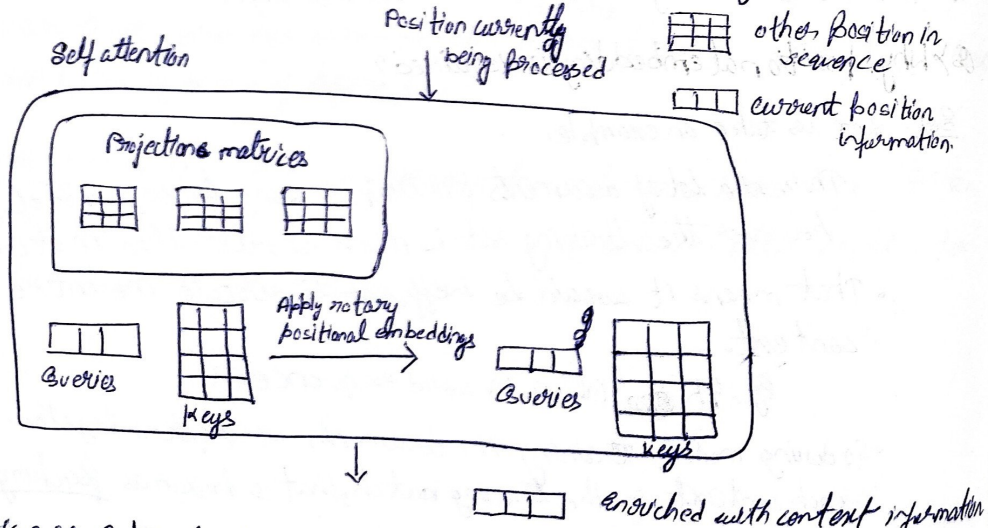


Fig 3.33. Rotary positional embeddings are added to the representation of tokens just before the relevance scoring step in self attention.

Q) "The dog chased the squirrel" use byte pair encoding to create vocabulary or find the no. of tokens.

Sol: Initial corpus : ["The", "dog", "chased", "the", "squirrel"]

Step 1 : Preprocessing

"The" : ["T", "he", "e", "_"]

"dog" : ["d", "o", "g", "_"]

"chased" : ["c", "h", "a", "s", "e", "d"]

"the" : ["t", "h", "e"]

"squirrel" : ["s", "q", "u", "i", "r", "r", "e", "l"]

Step 2: Iterative merging

1. Iteration 1

Most frequent pairs : ("t", "h", "e")

Merge into : "the"

corpus updates : ["The", "_"]

["d", "o", "g"]

["c", "h", "a", "s", "e", "d"]

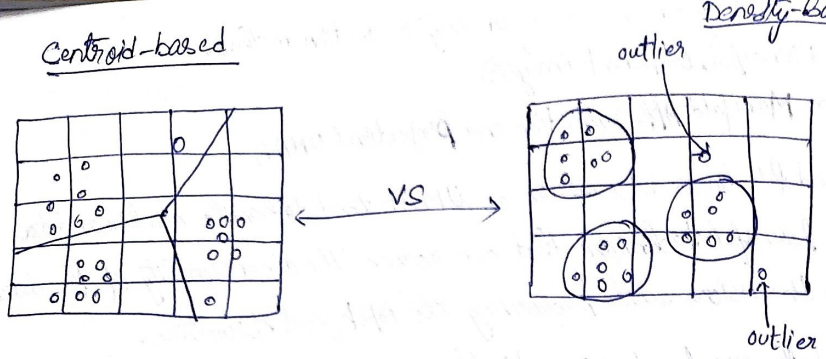


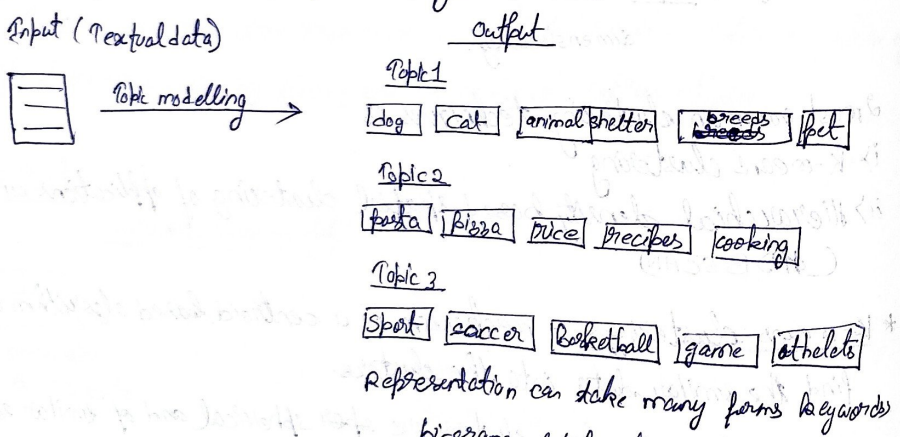
Fig: 5-7 The clustering algorithm not only impacts how clusters are generated but also how they are viewed.

- The HDBSCAN algorithm allows dense clusters ~~to be found~~ without having to explicitly specify the no. of clusters, allows only dense clusters.
- It also ~~decide~~ dictates the ~~outliers~~ outlier in the data which are the datapoints that doesn't belong to any ~~clus~~ cluster.

21/10

Topic modelling -

The process of providing themes on topics to a cluster according to their idea of finding themes each known as topic modelling.



Representation can take many forms keywords bi-grams, labels etc.

Fig: 5-9: Traditionally topics are represented by a number of keywords but can take another forms.

- Each topic is characterized by a probability distribution of words in a corpus vocabulary.

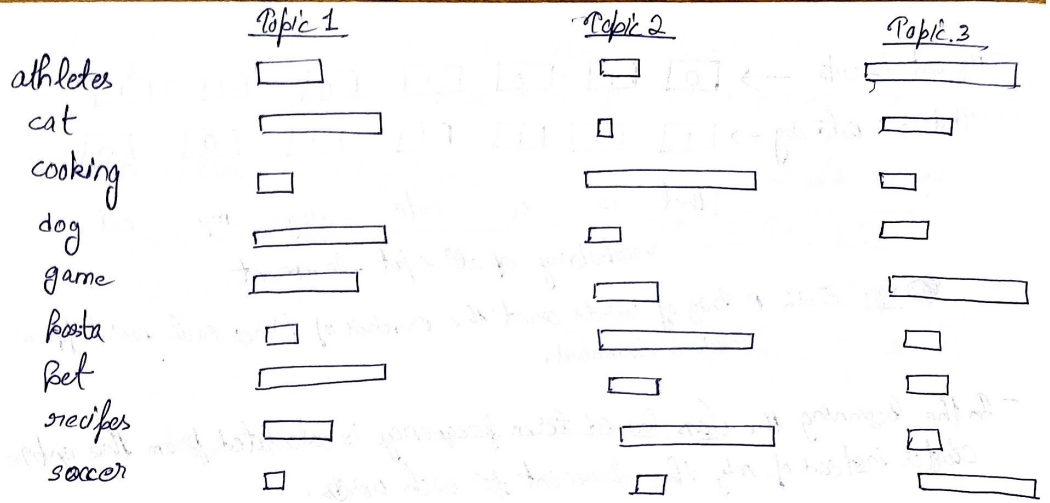


Fig: 5-10: Keywords are extracted based on their distribution over a single topic.

BERT Topic : A Modular Topic Modelling Framework -

- The BERTopic modeling leverage the clusters of semantically similar text to extract various types of topics representations.
- It follows two steps approach -

① Reduce the text dimensions after embeddings and cluster those compressed dimensions to create groups of semantically similar documents.

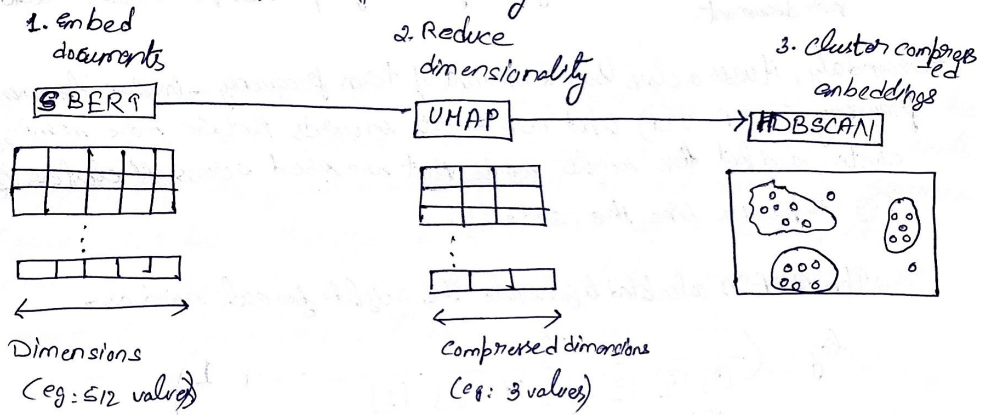


Fig 5-11: The first part of BERTopic's pipeline is to create clusters of semantically similar documents.

② BertTopic model provides a distribution of words in a corpus vocabulary by leveraging a classic method known as Bag of Words.

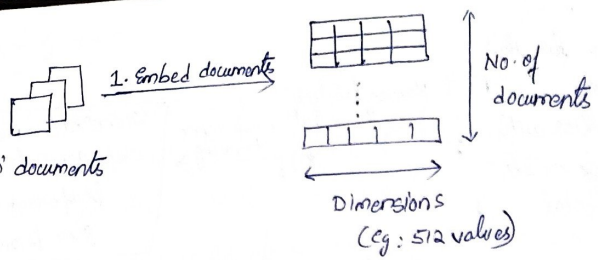


Fig 5.3 step 1 - We convert documents to embeddings using an embedding model.

- Here the textual data converting into embeddings means the numerical representations of text that attend to capture ~~a t~~ its meaning.

Step 2) Reduce the dimensionality of embeddings using dimensionality reduction model.

- The dimensionality reduction process can be troublesome for many clustering techniques when there have larger volume of data as it gets more difficult to identify the meaningful clusters.
- As the dimensions in terms of no. increases there is an exponential growth in the no. of possible values within each dimension.
- So, the dimension reduction technique aim to preserve the global structure of high dimension data by finding low dimensional representations.

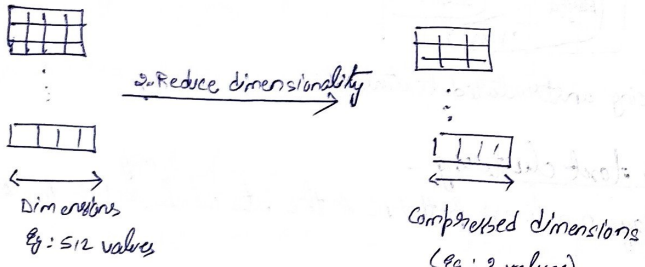


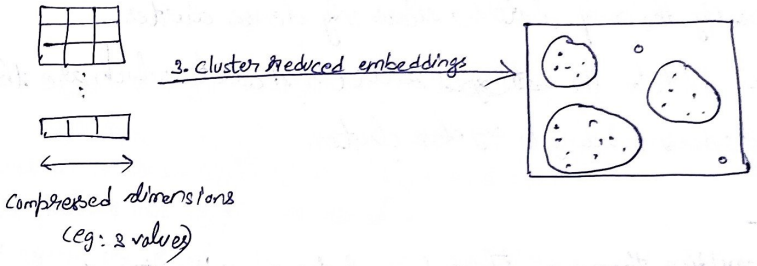
Fig: 5. step 2 : The embeddings are reduced to a lower-dimensional space using dimensionality reduction.

- There are 2 most common dimensionality reduction method.
  i) PCA (Principal Component Analysis)
  ii) Uniform Manifold Approximation and Projection (UMAP)

* PCA is used for linear data where as UMAP tends to handle non linear data.
* UMAP is a powerful technique that can reduce the dimensionality of the data into lower dimensions while preserving its topological structure.

Step 3) Find the groups of semantically similar ~~models~~ documents with a cluster model.

- Clustering of the documents requires a set of clusters to be generated.



~~Fig (Step 3)~~

Fig 5.6 step 3 : We cluster the documents using the embeddings with reduced dimensionality.

2 most common clustering techniques as
i) K-means clustering
ii) Hierarchical density based spatial clustering of applications with noise (HDBSCAN)

* K-means clusterings - mechanism is a centroid based algorithm used to find the similar data into the clusters.
  - It is widely used when clusters are ~~sper~~ spherical and of similar size.
* HDBSCAN - It is a hierarchical density based clustering method that calculates the no. of cluster to be found without ~~forch~~ forcing all datapoints to ~~be~~ be part of a cluster.

Text to text transfer transformer

- Instead of encoder only model (BERT) and decoder only model (ChatGPT), the text-to-text transfer transformer model (T5) contains both encoder and decoder for the classification of text.
- T5 model contains 12 encoder and 12 decoders that work for text-to-text transformers.

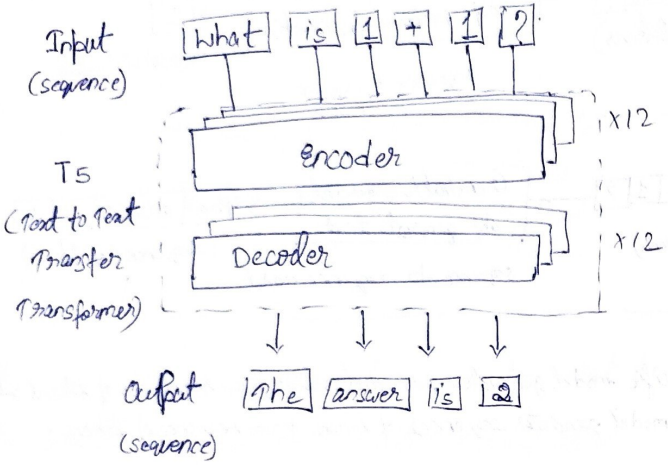


Fig: 4.19 The T5 architecture is similar to the original transformer model, a decoder-encoder architecture.

In 1st step,

- The models first pre trained with set of tokens known as masked language modelling.
- In 2nd step of training, the model is fine tuned with variety of task.

ChatGPT for classification -

- The ChatGPT uses open AI during the training process that generate the desired output to an input data in the prompt. every time it checks for the preference and final output will be accordingly preference-tunning.

For eg:

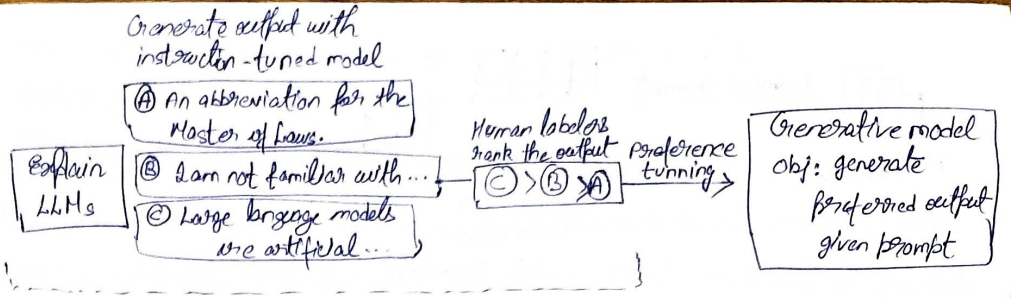


Fig: 4.23 Manually ranked preference data was used to generate the final model, chat GPT.

14/10

CHAPTER-5

Text clustering and topic modelling

Text clustering - is an unsupervised technique that aims to groups similar text based on their semantic content meaning and relationships.

- The resulting of clustering semantically similar documents not only fascinate efficient categorization of large volume of unstructured text but also allow for quick exploratory data analysis. (EDA)

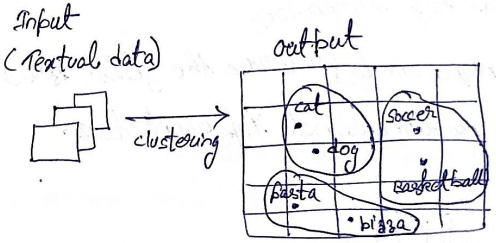


Fig 5.1 clustering unstructured textual data

A common pipeline for text clustering -

- Text clustering displays non linear patterns in the data which may contain the complexity in the analysis.
- To apply the text clustering from the graph based neural network to centroid based clustering technique with use 3 common steps and algorithms.

steps

1) Convert the input documents to embeddings with an embedding model.

Top_p - It is a nucleus sampling method which controls the subset of tokens (nucleus) that the LLM can handle. means it will consider tokens ~~and~~ until it reaches the cumulative probability.

For eg: If we ~~say~~ set top_p to 0.1, it will consider tokens until it reaches that value but if we say top_p to 1 then it will consider all tokens.

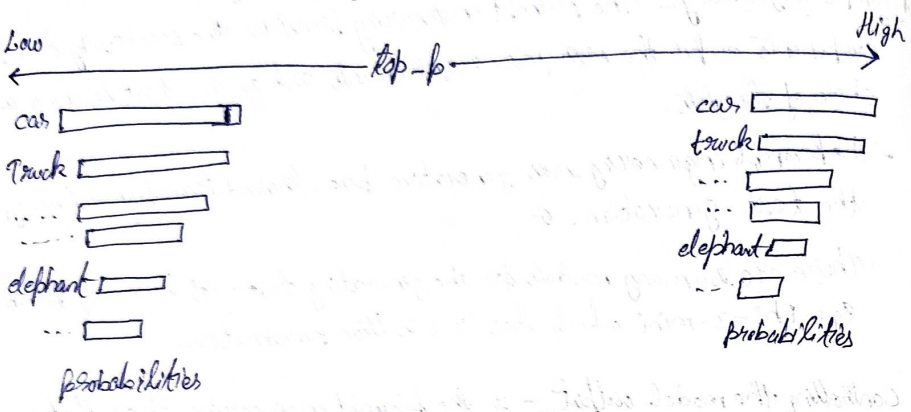


Fig 6.5 - A higher top_p increases the no. of tokens that can be selected to generate and vice versa.

In this figure, the lowering the value it will consider the fewer tokens and generally give less creative outputs while increasing the value the LLM to choose from more tokens.

Introduction to prompt engineering -

It is a tool that evaluate the output of a model as well as to design ~~safety~~ safety methods. This is known as Prompt ~~en~~ optimization.

- The basic LLM model is a prediction machine that is based on a ~~sub~~ certain inputs where the prompt tries to predict the words that might give the appropriate or ~~relate~~ relevant answers.

- In the prompt box we can ~~conti~~ continue adding or updating the elements until we can get the response what we are looking for.
- We can add additional examples ~~e~~.

Eg: Use cases in more details, provide additional context etc to get the result.

- In BERTopic the reranker models are known as representation models.
- For eg: If we have millions of documents and hundred topics then the representation model only needs to be applied once for every topic instead for every document.

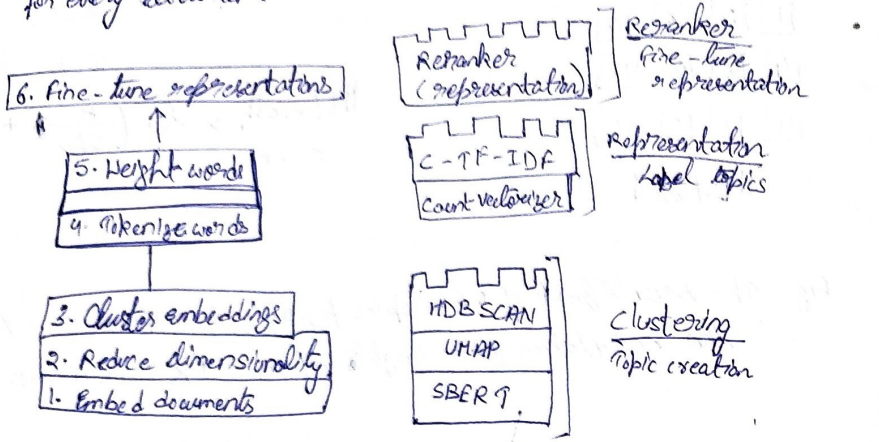
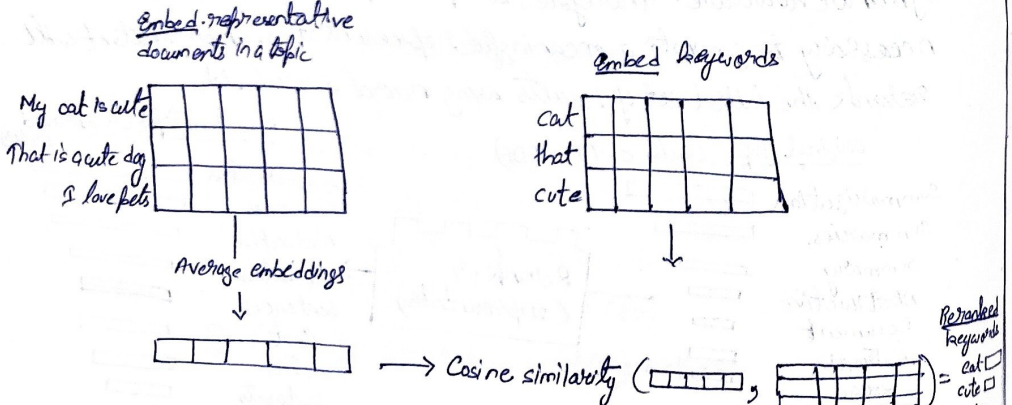


Fig 5.20 - The Reranker (representation) block sits on the top of c-TF-IDF representation.

- A wide variety of representation model have developed for.
  Eg: KeyBERTInspired
- KeyBERTInspired uses c-TF-IDF to extract the most represented documents per topic by calculating the similarity between a documents c-TF-IDF values and those topics related to that keywords.



5.22 KeyBERTInspired representation model

- In this figure the average document embeddings per topic is calculated and compare to the embeddings of candidate keywords to rerank the keywords.

4/11

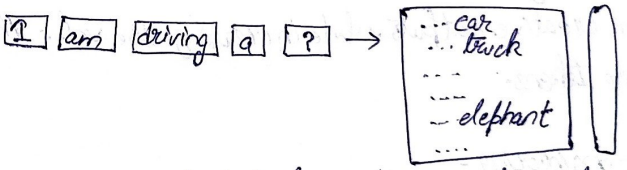
Chapter 6: Prompt Engineering

Prompt engineering - The prompt engineering provides the creativity and potential components as per the user requirements such as in context learning and chain of thoughts.

- The prompt engineering uses generative pre-trained transformer (i.e gpt) for the text generation.
- There are so many models for the generating text with Billion of parameters eg: Phi-3-mini which has 3.8 billion parameters.

Controlling the model output - In the prompt engineering the output can be generated according to the different response for the exact same prompt. each time an LLM needs to generate a token.

Eg: There is a sentence given as "I am driving a ?"



There are 2 important factors defined in the S function as
i) Temperature
ii) Top_p

- The temperature controls the randomness and creativity of the text that generated means it defines how likely it is to choose tokens that are less probable.
- It will generate the probability of any token with the same response every time bcz it always chooses the most likely word.

My cat is cute → [0] [1] [0] [1] [0] [1] [1]

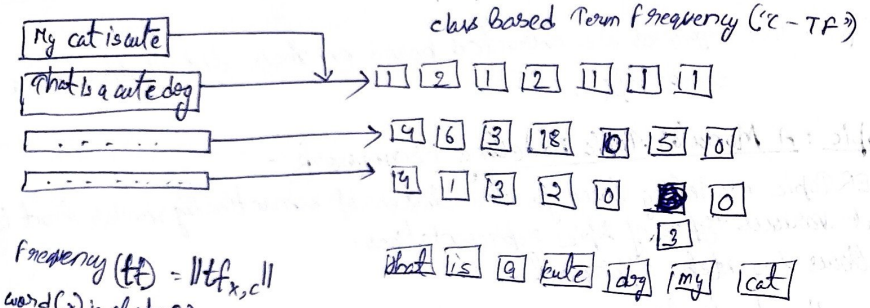
That is a cute dog → [1] [1] [1] [1] [1] [0] [0]

that is a cute dog my cat,

Vocabulary of all input document

Fig: 5-12 A Bag of words count the number of times each word appears inside a document.

- In the beginning the class based term frequency is calculated from the entire cluster instead of only the document for each words.



Frequency (tf) = $\|tf_{x,c}\|$
word(x) in cluster(c)

Fig - Generating c-TF by counting the frequency of words per cluster instead of per document.

- Secondaly, it uses a class based variant of term frequency - inverse document frequency (c-TF-IDF) to put more weights on words that are more meaningful to cluster and put less weights, words that are used across all clusters
(eg: stopwords like the, and, I).

The IDF is calculated to generate the weight for each word as -

$$\log\left(\frac{29}{[4]\,[9]\,[7]\,[22]\,[1]\,[9]\,[1]} + 1\right) = \log\left(\frac{A}{cf_x} + 1\right)$$

that is a cute dog my cat

Average frequency (A) divided by the total frequency of each word (cf_x)

Fig 5-14: Creating a weighting scheme

- In the last the weighted value c-TF-IDF is calculated by multiplying the result of a weight i.e IDF for each word with its frequency i.e c-TF.

4. Create a class-based bag-of-words

Count Vectorizer → [1 2 1 2] [4 6 3 18] [4 1 3 2]

$\|tf_{x,c}\|$

5. Weight terms

c-TF-IDF

$$\frac{\|tf_{x,c}\|}{c\text{-}TF} \times \overbrace{\log\left(\frac{A}{cf_x} + 1\right)}^{IDF}$$

Fig: The second part of BERTopics pipeline is representing the topics, the calculation of the weights of term "x" is a class "c".

24/10

Adding a special Lego Block - The pipeline in BERT topic has advantage like fast and modular but it follows some disadvantages as it still represents a topic through bag of words ~~with they are~~ without taking into ~~common~~ syntactic structure. Therefore, ~~to speed~~ speed up the process it is necessary to generate a meaningful representation. For that it will rerank the initial set of results using neural search like