Python

## Coverage :-

13 + End to End projects.

✓ ① Python programming - Basic to Advance

② NLP in Deep Learning → RNN, LSTM, GRU, RNN, Bidirectional RNN Encoder, Decoder, Attention Mechanism, Transformer, BERT.
(Above is Basic building block)
⇓
LLM models.

③ Langchain Ecosystem → Generative AI framework
⇓
Paid LLM, multimodal, Open Source LLM

Agents, Chat with SQL, vector Database, Retrievers, Text summerization
~~Chat message~~ chabot with chat msg history.

Langchain core
Langchain community
Langchain Tools

④ Deployment Techniques

⑤ Fine Tuning Techniques
Quantization, LORA QLORA.

⑥ GenAI in AWS : AWS BedRock, Lamda function.

⑦ Nvidia NIM

GoodLuck | Page No.
Date

* Streamlit

Streamlit is an open-source app framework for machine learning and Data science projects. It allows you to create beautiful web applications for your machine learning and data science project with simple python script.

```
st.title('Hello streamlit')  # To create a title for a page.
st.write(' ')  # To write simple text.
st.pline_chart(data)  # To create a line chart.
name = st.text_input('Enter your name: ')
# for taking input.
```

To run streamlit file we use command :
-> streamlit run filename.py

```
age = st.slider('select your age:', 0, 100, 25)
# creates a slider with defecault age 25.
# we can also create drop down select.
option = ['python', 'C++', 'Java', 'PHP']
choice = st.selectbox('Language', options)
file = st.file_uploader('choose a pdf', type='pdf')
# This will create an option to upload your document to the website.
```

Go to streamlit.io for more knowledge and components.

## * Machine learning for NLP

I/p text → vector | Text preprocessing 3 → word2vec, Text Avgword2vec

I/p text → vector | Text preprocessing 2 → BOW, TFIDF

Cleaning the I/p | Text preprocessing – Tokenization, Lemmatization, stemming

## * Tokenization in NLP

Tokenization is a process of converting the paragraph or a sten sentence into ~~simple~~ tokens.

- Corpus – paragraphs
- Documents – Sentences
- Vocabulary – unique words
- words – all the words pre

Natural Language Toolkit (
NLTK is a leading platform
program to work with hu
NLTK is free, ~~opp~~ open sour
driven.

- Sentence ← → paragraph
  → from nltk.tokenize import sent_tokenize
- Paragraph → word
  sentence → word
  → from nltk.tokenize import word_tokenize
  → wordpunct_tokenize
  → TreebankWordTokenize

## * Stemming

Stemming is the process of reducing a word stem that affixes to suffixes and prefixes or to the root of words known as a lemma.

- → PorterStemmer
  → from nltk.stem import PorterStemmer

- RegexpStemmer
  It is simple and more customizable stemming approach than PorterStemmer.
  It uses regular expression to define how to remove suffixes ~~back~~ from words.

  → from nltk.stem import RegexpStemmer
  a = RegexpStemmer('ing$|s$|e$|able$', min=4)

  Above RegexpStemmer is configure to remove suffixes like 'ing', 's', 'e' and 'able'.

- Snowball Stemmer
  It is a stemming algorithm which is also known as the Porter2 stemming as it is better version of Porter stemmer since some issues of it were fixed in this stemmer

## * Lemmatizer

Lemmatization technique is like stemming.
The output will get after lemmatization called as 'lemma', which is root word rather than ~~stem word~~ root stem.
It preserves the meaning of the word.

→ from nltk.stem import WordNetLemmatizer
a = WordNetLemmatizer()
a.lemmatize('going', pos='v')

# pos : Part of speech

~~non~~ noun – n
verb – v
adjective – a
adverb – r

GoodLuck | Page No. | Date

## * Stopwords

Stopwords are common words in a language that are usually filtered out before processing text data in NLP

These words are typically removed because they occur frequently and contribute little to the meaning or context of the text.

eg: 'a', 'the', 'is', etc.

→
```
Pmport nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stence sentence = 'This is a example'
words = nltk.word_tokenize(sentence)
a = [word for word in words if word.lower()
     not in set(stopwords.words('english'))]
print(a)
```

Output: ['example']

## * Part-of-Speech (POS) tagging

It is a process in NLP that involves labeling each word in a sentence with its corresponding part of speech such as noun, verb, adjective, etc. This is an essential step for understanding the grammatical structure of a sentence.

→
```
import nltk
sentence = 'The brown fox jumps over the
           fenz'
words = nltk.word_tokenize(sentence)
a = nltk.pos_tag(words)
print(a)
```

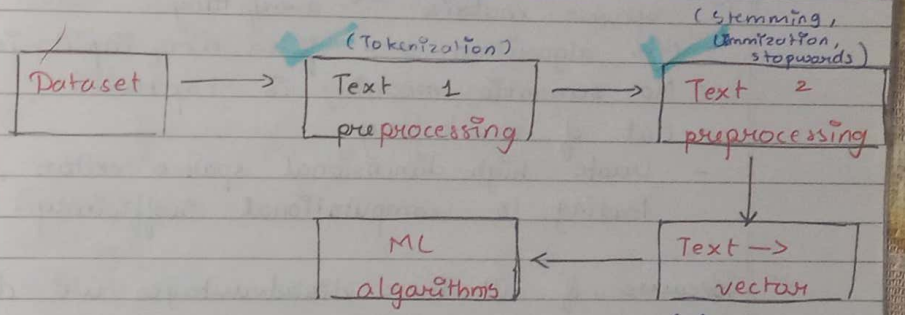
→ output: [('The', 'DT'), ('brown', 'JJ') etc.

GoodLuck | Page No. | Date

## * Name entity recognition (NER)

It is a crucial task in NLP that involves identifying and classifying name entities in text into predefined categories such as person names, organizations, locations, dates and more.

eg: Akanksha : person name
2003 : date.




(One hot encoding, Bag of words, TFIDF, word2vec, Avg word2vec)

## * One hot encoding

One-hot encoding is a fundamental technique in NLP used to represent words to vectors. Each word in a vocabulary is mapped to a unique word vector, where only one element is set to 1 and all other elements to 0.

Example: vocabulary of words: ['cat', 'dog', 'fish']

cat = [1, 0, 0]
dog = [0, 1, 0]
fish = [0, 0, 1]

Advantage :-
- Easy to implement with python.
- It does not make any assumptions about the relationship between words.
- Clearly distinguishes between words.

Disadvantages :-
- sparse matrix → overfitting
- ML algorithm → fixed size input is required
- No semantic meaning is captured.
- Out of vocabulary
- Create high-dimensional sparse vector, leading to computational inefficiency.

Because of all these disadvantages we don't use one hot encoding in NLP.

★ Bag of words

Bag of words is ~~another~~ a fundamental technique in NLP used for text representation. It converts text into numerical features.

Example:

1. He is a good boy.
2. She is a good girl.
3. Boy and girl are good

Stopwords → Lower all the words case

| | Vocabulary | frequency |
|---|---|---|
| S1 → good boy | good | 3 |
| S2 → good girl | boy | 2 |
| S3 → Boy girl good | girl | 2 |

| | good | boy | girl |
|---|---|---|---|
| S1 | 1 | 1 | 0 |
| S2 | 1 | 0 | 1 |
| S3 | 1 | 1 | 1 |

^ Bag of words (BOW)

There are 2 types of BOW :-
- Binary BOW = 1 & 0
- BOW = count will get updated based on frequency.

Advantages :-
- Simple and intuitive
- Fixed sized I/p — machine learning algorithms

disadvantages :-
- Sparse matrix - overfitting
- Ordering of the word is getting changed.
- Out of vocabulary
- Semantic meaning is still not captured.

★ TF - IDF

Term Frequency - Inverse Doument Frequency

It reflects the importance of a word in a document.

Term frequency = $\frac{\text{frequency of a word in a document}}{\text{total no. of words in document.}}$

Document → sentence.

$$IDF = \log_e\left(\frac{\text{Total no. of documents in corpus}}{\text{no. of documents containing word}}\right)$$

TF-IDF = TF * IDF

Example:-

S1 = good boy
S2 = good girl
S3 = boy girl good

Term-frequency

| | $S_1$ | $S_2$ | $S_3$ |
|---|---|---|---|
| good | 1/2 | 1/2 | 1/3 |
| boy | 1/2 | 0 | 1/3 |
| girl | 0 | 1/2 | 1/3 |

Inverse-document frequency

| | IDF |
|---|---|
| good | $\log_e(3/3) = 0$ |
| boy | $\log_e(3/2)$ |
| girl | $\log_e(3/2)$ |

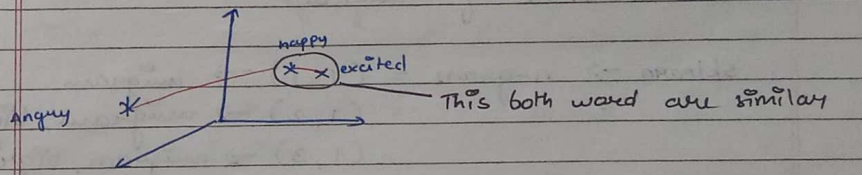
$$\therefore \text{TF-IDF} = \begin{matrix} S_1 \\ S_2 \\ S_3 \end{matrix}\begin{bmatrix} \text{good} & \text{boy} & \text{girl} \\ 0 & 1/2 \times \log_e 3/2 & 0 \\ 0 & 0 & 1/2 \log_e 3/2 \\ 0 & 1/3 \log_e 3/2 & 1/3 \log_e 3/2 \end{bmatrix}$$

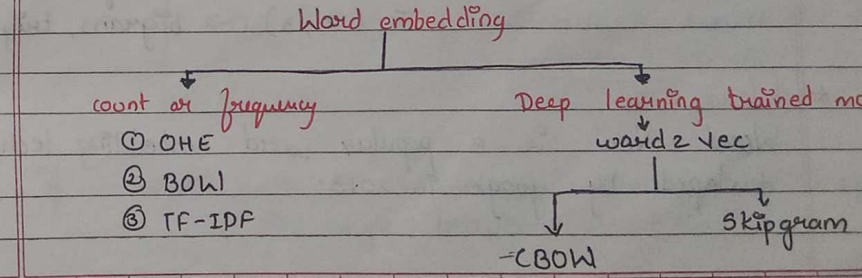
Advantage
- Intuitive : easily understanable
- Fixed size input
- Word importance is getting captured.

Disadvantage
- Spares matrix.
- Out of volcabulary

* Word embedding
It is technique used in NLP to represent words as vectors in continuous vector space.
This allows words with similar meanings to have similar representations.
Words that are closer in the vector space are expected to be similar in meaning.

Example:
If we represent the words: angry, happy and excited, in the vector space:-



★ ~~Word 2 Vec~~

★ N-grams

An n-gram is a sequence of 'n' items from a given text or speech.

- Unigram (1-gram)
  Sequence of ~~single~~ with single item
  eg: I love NLP
  unigram : ['I', 'love', 'NLP']

- Bigram (2-gram)
  A sequence of two items
  eg: I love NLP
  bigram : ['I love', 'love NLP']

- Trigram (3-gram
  A sequence of 3 times
  eg: I love NLP
  trigram : ['I love NLP']

N-gram is essential for capturing the context and structure of language.

sklearn → n-gram = (1,1) → unigram
= (1,2) → unigram, bigram
= (1,3) → unigram, bigram, trigram
= (2,3) → bigram, trigram.

★ Word 2 Vec

Word 2 Vec is a popular word embedding technique developed by google in 2013.



It is designed to ~~capture~~ transform word into continuous vectors that capture semantic meanings and relationships between words. Word 2 Vec creates these vectors by using a shallow, two-layer neural network.

Once trained such models can detect synonymous words or suggest words for partial sentence.

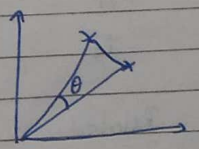
Each and every word present in the vocabulary is converted into feature representation.

↓

means converting all the words in vectors based on some features.

| Features | vocabulary: boy | girl | king | queen | apple |
|---|---|---|---|---|---|
| Gender | -1 | 1 | -0.92 | 0.93 | 0.01 |
| Royal | 0.02 | 0.05 | -0.95 | 0.96 | . |
| Age | . | . | . | . | . |
| Food | . | . | . | . | . |
| : | | | | | |

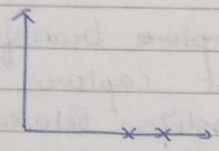
- Cosine similarity




Distance = 1 - cosine sim

cosin-sim = cosθ

where, x → represents word vector in space
θ → angle between word vectors



Distance = 1 - cos θ°
= 1 - 1
= 0

Thus words are similar.
or same words
In this way recommendation is done.

word2Vec
↓ CBOW
↓ Skipgram

* CBOW (Continuous Bag of Word)
CBOW is a fully connected neural network.
This model predicts the current word based on its surrounding words.
Suppose we have corpus :- E
[iNeuron Company is Related To Data Science]

and we give window size = 5
window size indicates how many words to be selected initially.

| I/P | O/P |
|---|---|
| iNeuron, Company, Related, To | is |
| Company, is, To, Data | Related |
| is, Related, Data, Science | To |

window size = 5 → indicates
when a word is converted to vector, will we get a size of five dimension vector which will our final output
Window size represents basically the feature representation of a word.

* Skipgram
This model predicts the surrounding context words for a given target word.

| I/P | O/P |
|---|---|
| is | iNeuron, company, Related, To |
| Related | company, is, To, Data |
| To | is, Related, Data, Science |

Small dataset → CBOW
Huge dataset → skipgram

- Advantages
  - Spare Matrix → dense matrix.
  - Semantic information is capture.
  - Vocabulary size
  - Out of vocabulary solved.

GoodLuck | Page No. | Date

## NLP In Deep Learning

This is prerequisite for Generative AI

Simple RNN → LSTM / GRU RNN

↓

Encoder-Decoder ← Bidirectional RNN

↓

Self attention → Transformers.

→ To maintain the sequential information, ~~pass~~ pass one word at a time instead of passing the entire sentence at once (like in ANN)

ANN : providing all the information then we do forward propogation and then backward propogation

We don't use ANN for Textual data because:-

→ It doesn't maintain the order (sequence) of the data, and because of that information loss takes place.

→ It passes the entire input sequence at once to the network instead of word by word.

## Simple RNN

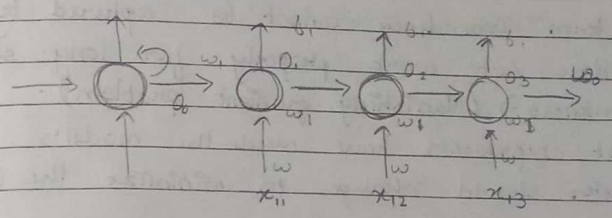
Type of neural network designed to process sequential data.

It introduces a feedback loop, allowing information to persist across time steps.

GoodLuck | Page No. | Date

## Forward propogation

$x_{11}$ → word 1 of sentence 1




$O_1$ = output

$w_1$ = hidden weight of neuron

Forward propogation

$O_1 = f(x_{11} \cdot w + b_1)$

$O_2 = f(x_{12} \cdot w + O_1 w_1 + b_1)$

$O_3 = f(x_{13} \cdot w + O_2 w_1 + b_1)$ → sent to activation function → $\hat{y}$ (final output)

## Backward propogation

After this ~~.~~ & calculation of final output there will be some loss

$$loss = y - \hat{y}$$

So main goal is to reduce this loss

So we can do this by updating the weights $[w, w_1, w_0]$

So during backward propogation these weights will be updated. and then again forward propogation is performed.

Backward propagation with time

Update $[w, w_1, w_0]$

weight update formula,

$$w_{new} = w_{old} - \eta \frac{\partial loss}{\partial w_{old}}$$

← Derivate, slope of gradient descent

All the 3 weights are updated using this above formula.

GoodLuck | Page No. | Date

★ Problems with RNN

- Long term Dependency cannot be captured by RNN (It does not work properly for long ~~sten~~ sentences.) (Vanishing gradient problem).

Gradient represents how much the model's parameter should change to minimize the loss function.

In backpropagation, the gradients are calculated layer by layer, starting from the output layer and moving backward to input.

During this the weights till ~~for~~ input becomes equivalent to ≈0. thus not having much effect on the output.

★ LSTM RNN

Long - short term memory RNN solves the problems regarding simple RNN

- It is designed to remember information for long periods. (It's like a smart brain cell that can decide what to remember or forget).
- LSTM achieve long term memory by using a memory cell and gates to control information flow.

★ Basic Architecture

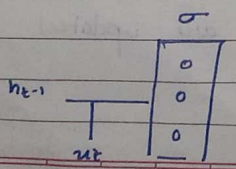
Basic termonologies :-

-

: Neural network layer

This is the way to represent Neural Networ layer in an architecture.

meaning :-




a Neural network and a sigmoid function is applied to it.

GoodLuck | Page No. | Date

- ⊙ : Pointwise operation

Example : $Y_1 = \begin{bmatrix}1\\4\end{bmatrix} \begin{bmatrix}2\\5\end{bmatrix} \begin{bmatrix}3\\6\end{bmatrix}$

⊗ => [4 10 18]

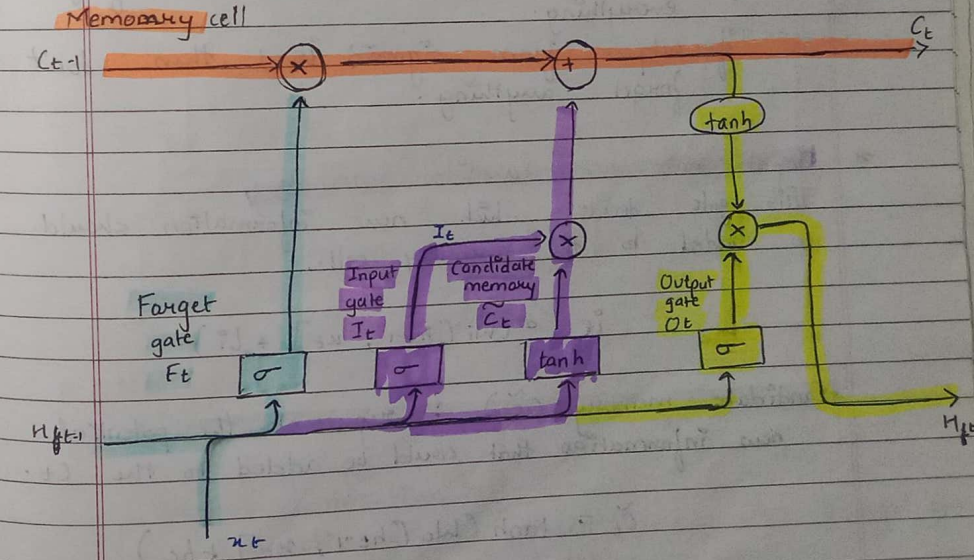
⊕ => [5 7 9]

- → Vector transfer

- ≻ Concatenate

Basically combining 2 vectors

- ⋋ Copy




★ Memory cell

Long - ~~short~~ term memory ~~who~~ whatever info. we want will be store and whatever is not required is removed.

(Short term memory)

- $h_{t-1}$ : hidden state of previous timestamp.
- $x_t$ : word passed as input in current timestamp.

★ **Forget gate** ($f_t$) :

Here in input ($x_t$) is taken and concatinated ($\rightarrow$) with & hidden layer ($h_{t-1}$), then passed through a sigmoid activation function ($\boxed{\sigma}$) and then lastly cross producted ($\otimes$) with previous long-term memory ($C_{t-1}$)

- It is used to forget information from memory cell based on the context.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- sigmoid value ranges between 0-1
- if output from sigmoid is 0 then forget everything.
- if output from sigmoid is 1 then don't forget anything.

★ **Input Gate and candidate memory**

This gate decide which new information should be added to the memory cell.

$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i)$$

- Candidate memory ($\tilde{C}_t$) : represents the potential new information that could be added to the $C_t$.

$$\tilde{C}_t = \tanh(W_c [h_{t-1}, x_t] + b_c)$$

Candidate memory : new notes to be added.
Input gate : how much notes should be added

The input gate decides how much of the candidate memory should be stored in cell state.

★ Updation after use of forgate gate & input gate.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$C_t$ : long term
$f_t$ : forget gate
$i_t$ : Input gate

★ **Output gate**

This gate decides what the next hidden state should be;
Calculation of $h_t$.

$$O_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

∴ Hidden state

$$h_t = O_t * \tanh(C_t)$$

$h_t$ : short-term

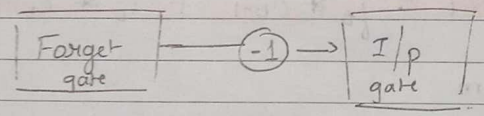
- Variants of LSTM.

1. Peephole connections used in LSTM

Peephole connection ↔
Connection from memory cell to Forget gate, input gate and output gate.

$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$O_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

2. Coupling forget and I/p gate.
Instead of seperatly deciding what to forget and what we should add new input input, we make this decision together.

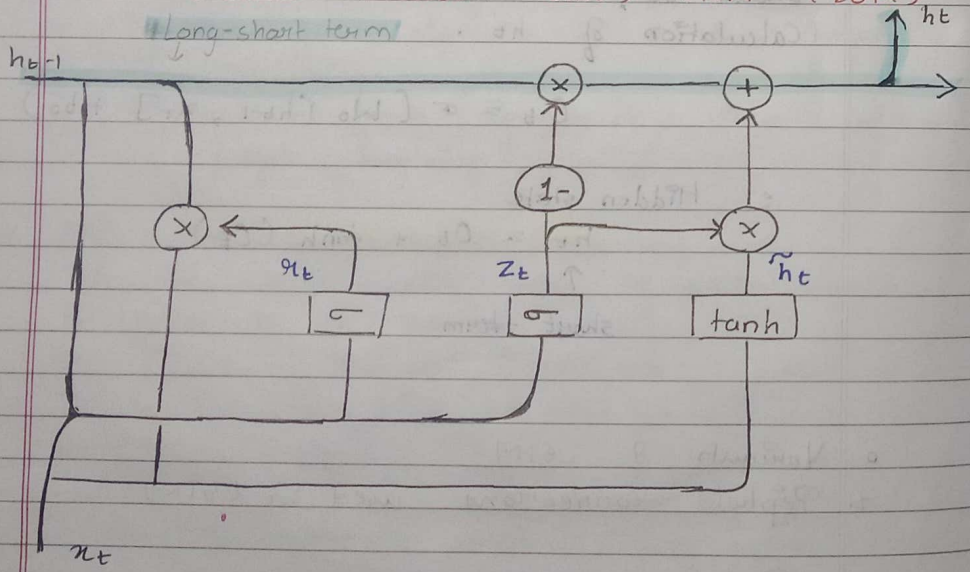



- We only forget when we're going to input something in it place and vice versa.

$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

3. GRU (Gated Recurrent Unit) RNN (2014)

Long-short term




$z_t$ : Update gate
$r_t$ : Reset gate
$\tilde{h}_t$ : temporary hidden state.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Reset gate ($r_t$) : reseting some information from $h_{t-1}$

- Update gate :
What context info needs to be added.
⇓ Depends on
(Temporary) candidate hidden state [current context]

Controls how much of the privous hidden state to keep and how much of new state to update with.

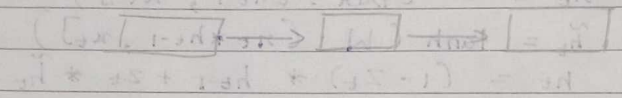
* Bidirectional RNN
When we need context of past words as well as further words there we use bi-RNN
example : I eat ___ in Banglare
- so here I need the context of both privious as well as further words to predict the word.

Bi-RNN consists of two separate RNNs :-
- One RNN processes the sequence from start to end.
- Another RNN processes the sequence from end to start.

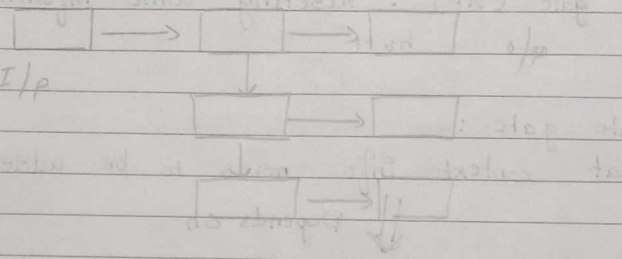
GoodLuck | Page No. | Date

* Types of RNN:

- One to One

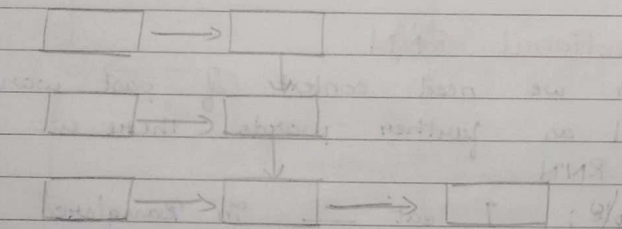


- One to many



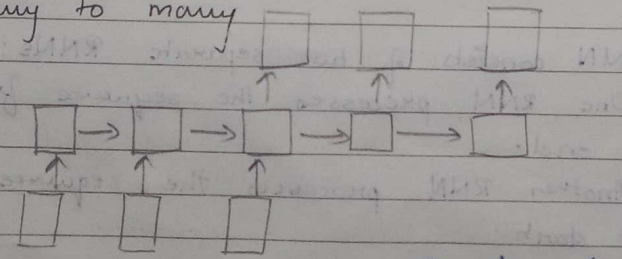

example : Image capturing
google image : we provide a image and it tell us what that image is.

- Many to one



Example : Image search.

- Many to many

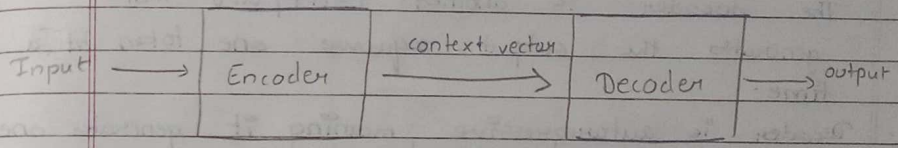


Example : Languag translation.

GoodLuck | Page No. | Date

* Encoder and Decoder

- The encoder processes the input sequence one element at time.
- It converts the sequence into a fixed length context length
- It is used to solve seq^ to seq^ problems.

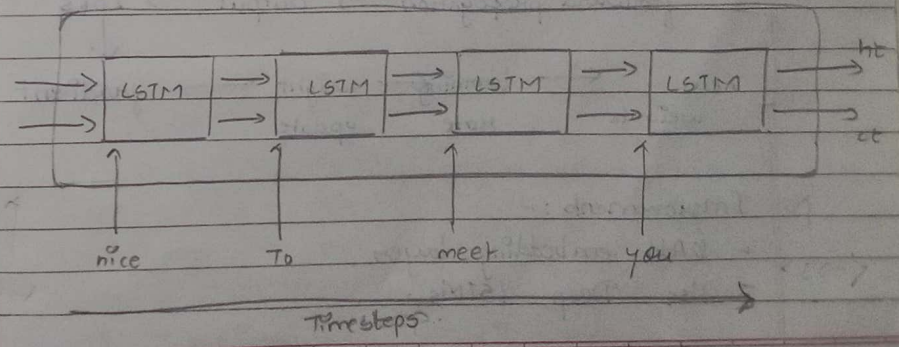



A High Level Overview

The encoder - decoder is a fundamental neural network commonly used in seq2seq tasks such as machine translation, text summarization, etc.

Encoder - Decoder architecture consists of two main components:-

* Encoder
- Processes the input sequence and generates a context vector.
- Typically implemented using LSTM or GRU cells.
- Example English → hindi translation.



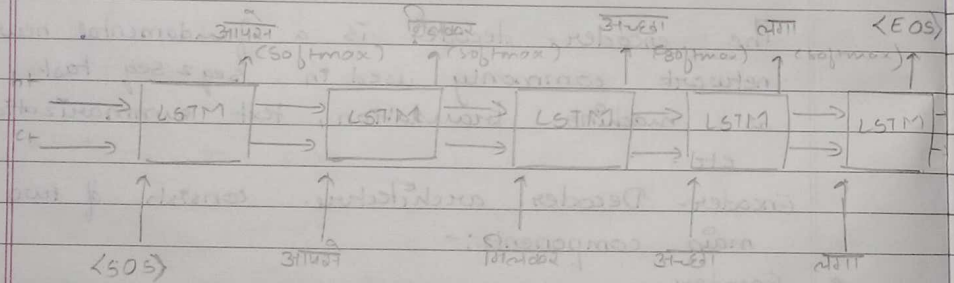

A encoder

- Context Vector
  - The last hidden state from the encoder acts as summary of the entire sequence.
  - It is passed to the decoder as its initial state.

* Decoder

The decoder is another LSTM / GRU that generates the output sequence one token at a time.

- Decoder is autoregressive, meaning it generates one token at a time and use ~~prev~~ previously generated tokens to predict the next token.




Decoder

- Training :-

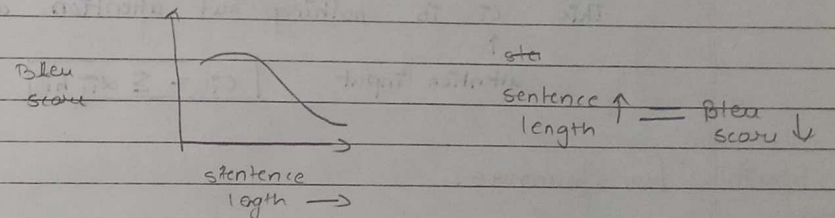
forward propogation → output → Loss

new weights ← learning rate ← weight update ← gradient

- Improvements :-
  - Add embedding layer
  - Use Deep LSTMs.

- Limitations
  - Fixed-length context vector. (Bottleneck Issue)
    If the input sequence is long, this context vector cannot retain all important details, leading to information loss.
  - Requires large Datasets for good performance.

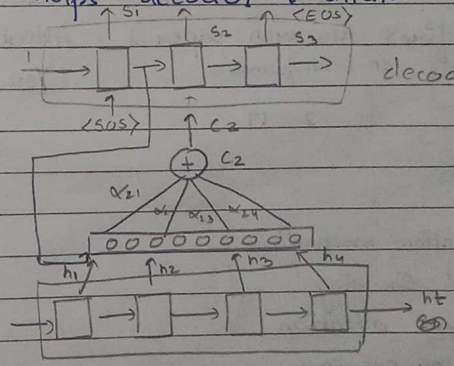
BLeu score : performance metrix.




* Attention Mechanism

It helps models focus on the most relevant parts of the input when generating output.

- It solves the bottleneck issue in Encoder-decoder by allowing the decoder to access all input token dynamically, instead of relying on a single fixed-length context vector.
- Attention helps decoder 'attend' to specific words




$\alpha_{21} = f(h_1, s_1)$

$s_1$ → previous hidden state of decoder.

$C_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3 + \alpha_{24} h_4$

(vanilla)

In normal encoder decoder, we usually only take $s_i$ and a context vector to calculate the next word in decoder.

But in attention mechanism new variable is added $C_i$ with $s_i$ and context vector (ht)

This $C_i$ is nothing but attention 'attend'.

↑ attention input

$$C_i = \sum \alpha_{ij} h_i$$

## * Transformers.

Attention mechanism (limitation) - Parallely we cannot send all words in a sentence (they were sent one by one).

- Not scalable : enable to handle large datasets

Transformers ✳→ LSTM RNN

↳ Self attention module

All the words will be parallely sent to the encoder.

Paper (first research paper) : Attention ~~all~~ Is All You Need

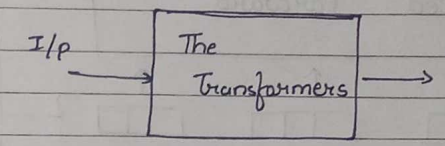
on transformers

2017

Features :-

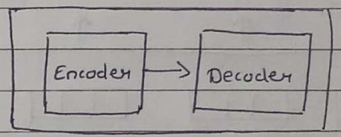
- Self-attention mechanism
- Positional Encoding
- Multi-head attention
- Parallel processing.
- Feed forward Neural Network.

## * Basic Transform Architecture



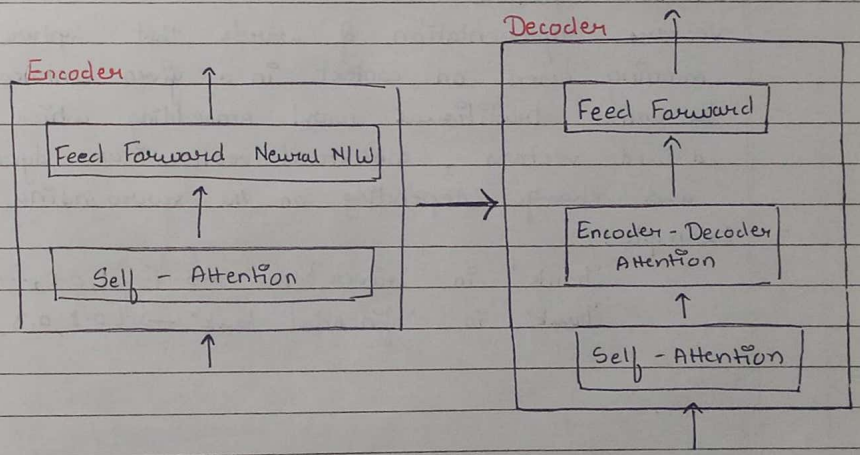

↓ go inside this transformer

In research paper 6 E-D were specified.

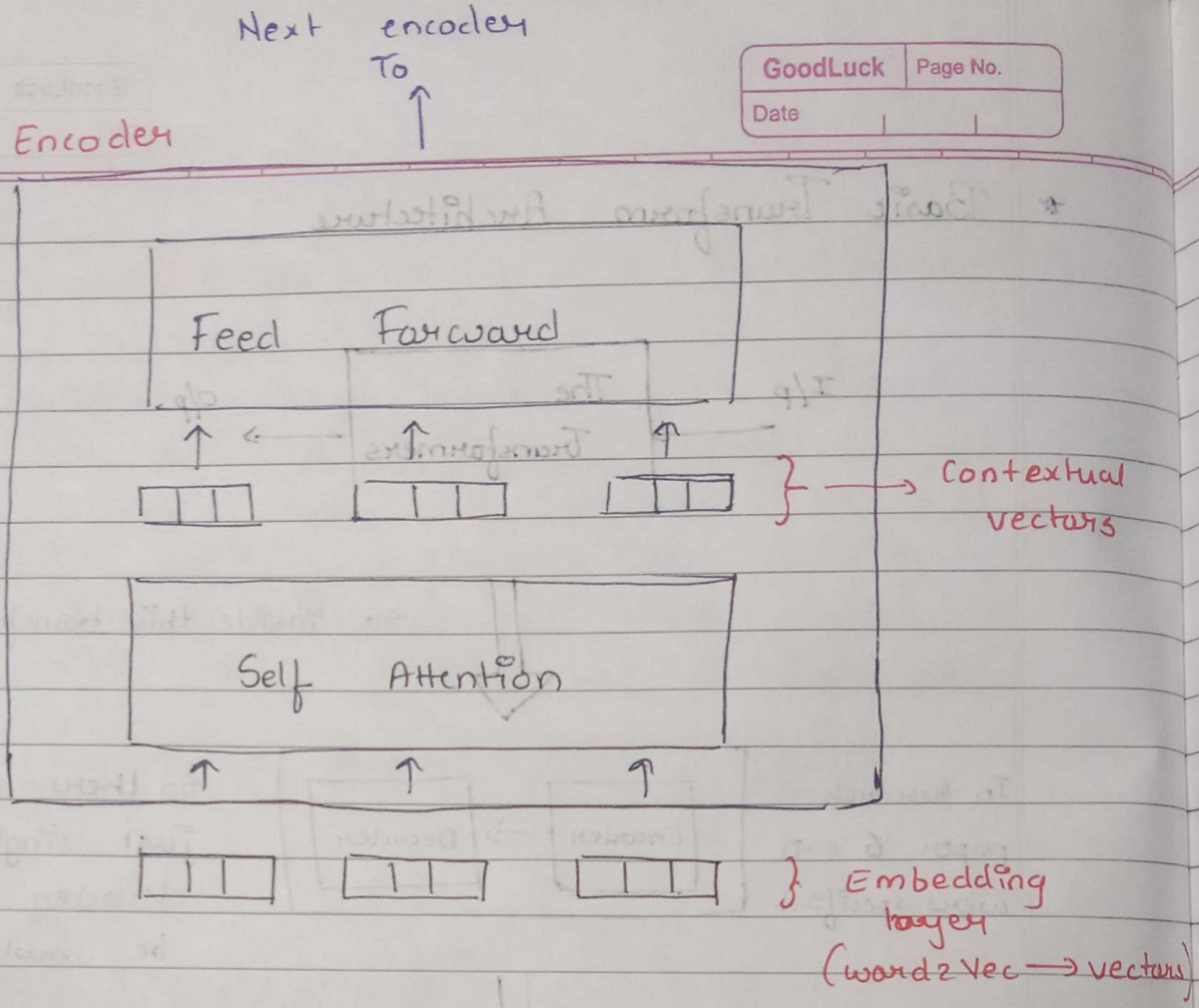



So there won't be just single encoder decoder, their will be multiple encoder-decoder

↓ so what is inside encoder and decoder.




↓ Let's look into encoders.




★ Contextual vectors

- Vectors representation of words that capture their meaning based on context in a given sentence.
- Unlike traditional word embedding which assigns a fixed vectors, contextual vectors are dynamic and change depending on the surrounding words.
- Example:

'bank' in 'river bank' → [0.2, 0.5, ...]

'bank' in 'financial bank' → [0.8, 0.3, ...]

Page No.
Date

★ Self Attention at higher and detailed Level

Self-attention, also known as scaled dot-product attention, is a crucial mechanism in the transformer architecture that allows the model to weigh the importance of different tokens in the input sequence relative to each other.

- Creates contextual vectors.

Steps :-

1) Token Embedding

Let's take example

Input sequence = ['The', 'cat', 'sat']

Embedding size = 4

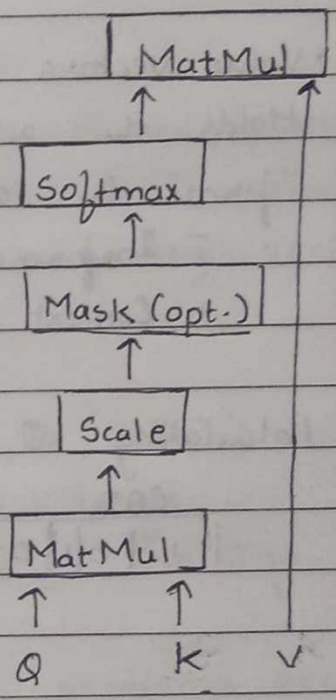
Q, K, V = 4

$E_{The} = [1\ 0\ 1\ 0]$

$E_{cat} = [0\ 1\ 0\ 1]$

$E_{SAT} = [1\ 1\ 1\ 1]$

Scaled Dot-Product.




2) Linear Transformation

We create Q, K, V by multiplying the embedding by learned weights metrics $W_Q$, $W_K$ and $W_V$.

- Query (Q): represents the token for which we are calculating the attention. They determine the ~~context of~~ importance of ~~current~~ other tokens in the context of the current token.
  - Focus Determination
  - Contextual Understanding

- Key Vector (K)

Represents all the tokens in the sequence and are used to compare with query vector to calculate attention score.

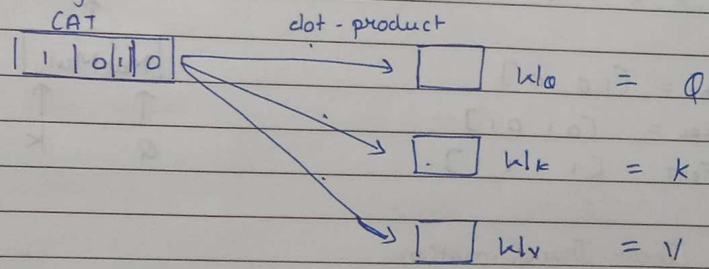
- Relevance measurement : compatibility of each token with the current token.
- Information retrieval

- Value vectors (V)

Holds the actual information that will aggregate to form the output of the attention mechanism.

- Information aggregation
- Context preservation.

Calculating Q, K, V :-




so for each token (word) these 3 vectors are calculated.

3) Compute attention Scores

By calculating the dot product between a query vector and all the key vector, the model assesses how much attention to give to each token relative to the current token.

Example :-

The

Score ($Q_{The}$, $K_{The}$) = $[1\ 0\ 1\ 0] \cdot [1\ 0\ 1\ 0]^T = 2$

Score ($Q_{The}$, $K_{cat}$) = $[1\ 0\ 1\ 0] \cdot [0\ 1\ 0\ 1]^T = 0$

Score ($Q_{The}$, $K_{SAT}$) = $[1\ 0\ 1\ 0] \cdot [1\ 1\ 1\ 1]^T = 2$

Similarly we calculate for ~~the~~ rest of the tokens

4) Scaling

We take up the score and scale them down by dividing the score by $\sqrt{d_k}$ (dimension of key vector), so the here dimension of key vector is 4 $\therefore \sqrt{4} = 2$.

Scaling in the attention mechanism is crucial to prevent the dot product from growing too large.

⇒ Ensure stable gradients during training.

If we don't do scaling following problems occur :-

1. Gradient Exploding
2. Softmax saturation.
   ↳ most of the weight is assigned to a single token and other tokens weight will be assigned near to zero.

- Property of softmax
  - Softmax [(6, 4)] = ≈ (0.88, 0.12)

most of the attention weight is assigned to the first key vector and very little to second vector.

- Softmax [(10, 1)] = ≈ (0.99, 0.01)

- Now with scaling.

[6, 4] ⇒ scale ⇒ [6/2, 4/2] = [3, 2]

softmax([3, 2]) ≈ (0.73, 0.27)

Here, the attention weights are more balanced compared to unscaled case.

- It stabalize Training.
- Preventing saturation.

Scaled-score $(Q_{The}, K_{The}) = 2/2 = 1$

$(Q_{The}, K_{cat}) = 0/2 = 0$

$(Q_{The}, K_{sat}) = 2/2 = 1$

Why scaling will be done to all other tokens.

5) Apply softmax

Attention weights 'The' = softmax([1, 0, 1])

= [0.4223, 0.1554, 0.4223]

Attention weights 'cat' = softmax([0, 2, 2])

= [0.1554, 0.422, 0.422]

Attention weights 'sat' = softmax([2, 2, 4])

= [0.219, 0.219, 0.472]

$$\text{Softmax}(x, y) = \left[\frac{e^x}{e^x+e^y}, \frac{e^y}{e^x+e^y}\right]$$

6) Weight sum of values

We multiply attention weight by corresponding value vector.

Output 'THE' = 0.4223 * $V_{THE}$ + 0.1554 * $V_{CAT}$ + 0.4223 * $V_{SAT}$

= [1.2669, 0.9999, 1.2669, 0.9999]

↑ attention head (z)

Why for other tokens are calculated.

The → [1 | 0 | 1 | 0] ⇒ Attention Self ⇒ [1.2669, 0.9999, 1.2669, 0.9999]

Steps in short.

1. Calculated Q, K, V
2. Attention score
   - scaled
   - Softmax
   - weighted sum of values

★ Multi-Head Attention

While calculating Q, K and V if we change ours weight values to different than previous ones we will get the final output (z) as different.

This nothing but multi-headed attention

- It expands model's ability to focus on different position of tokens.

So from our self attention we generate multiple attention heads (Z1, Z2, Z3, etc.)

So now how do we combine all these things before giving it to the feed forward neural network.
So for that :-

1. Concatenate all the attention heads
2. Multiply with the weight matrix W° that was trained
3. The result would be the Z matrix that captures information from all the attention heads.

Now we can send this to Feed-Forward Neural Network.

★ Positional Encoding

So upto now we are processing the words parallely ⟹ Lack of sequential structure of drawback

the words (order). So how do we know or encoder knows the order of the words, so here comes the role of positional encoding.

- Representing order of sequence.

We create a positional encoded vector and add it to embedded vectors, so that will give the position information.

Type of positional encoding :-

1) Sinusoidal Positional encoding

It uses sin and cosin functions of different frequencies to create positional encoding.

Formula :-

$$P.E(position, 2i) = \sin\left(\frac{pos}{10000^{2i/dmodel}}\right)$$

$$P.E(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/dmodel}}\right)$$

where,

pos / position = position of the word.
i = dimension,
dmodel = dimensionality of the embedding

Example :-

| | THE | CAT | SAT | |
|---|---|---|---|---|
| i = | 0 | 1 | 2 | 3 |
| 0 THE → | [0.1 | 0.2 | 0.3 | 0.4] |
| 1 CAT → | [0.5 | 0.6 | 0.7 | 0.8] |
| 2 SAT → | [0.9 | 1.0 | 1.1 | 1.2] |

so dmodel = 4
pos of cat = 1 i = iterates of over 0 to 3
pos of THE = 0
pos of SAT = 2

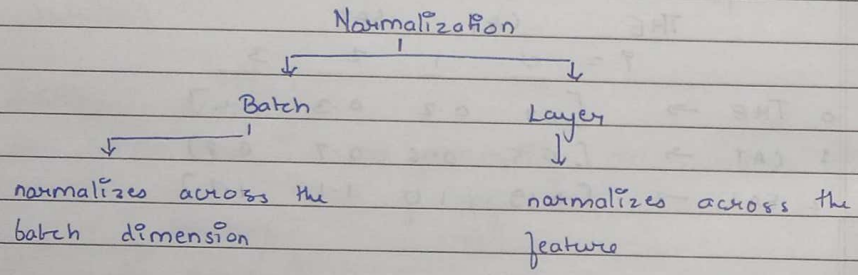
so even numbers ~~position~~ i will use sin formula
odd i will use cos formula

We do not concatinate the positional vector with embedding vector because dimensional will increase that's why we add up.

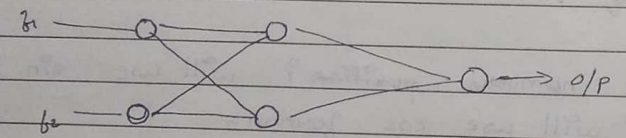
## ★ Layer Normalization

Before passing our contextual vector to feed forward it is (ADD & Norm) Normalized using layer normalization.

Normalization is a technique used to stabilize and standardize the input data and activations at each layer of transformer model.
It typically involves normalizing the values so that they fall within a standard range, often between 0 and 1 or -1 and 1.




There are various issues with batch normalization that's why we use layer normalization.




Normalization

$$z = \frac{x_i - \mu}{\sigma}$$

$\mu = 0$

$\sigma = 1$

1. Compute mean & variance across the features.

2. Normalize the input

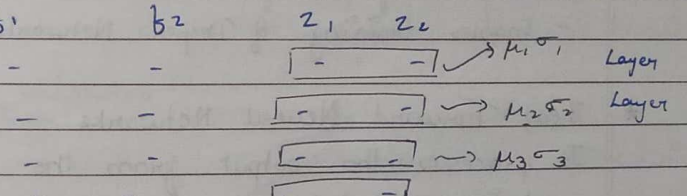
$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

where $\epsilon$ = small constant to prevent division by zero.

3. Scale and shift

Apply learnable parameters $\gamma$ (scale) and $\beta$ (shift) to maintain the model's ability to represent complex functions:

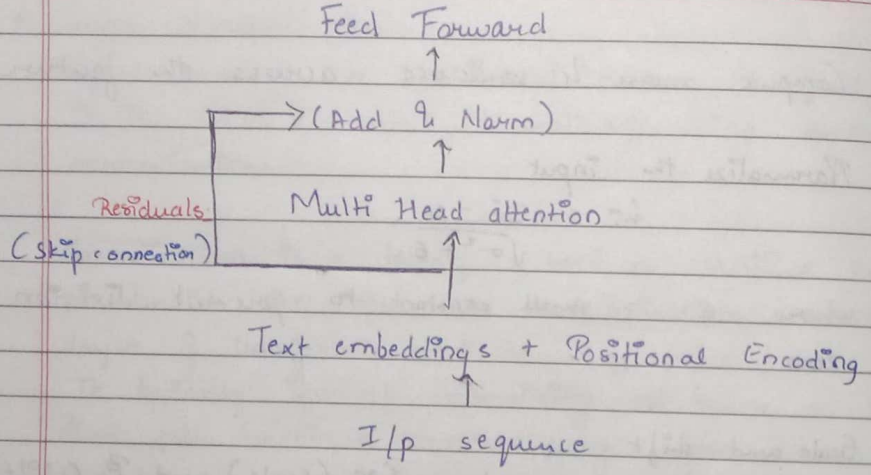
$$y_i = \gamma \hat{x}_i + \beta$$




$$z_{score} = \frac{x_i - \mu_1}{\sigma_1} \qquad z_{score} = \frac{x_i - \mu_2}{\sigma_2}$$

$$z_1 = \sigma[w_i^T x + b_1]$$

$$y = \gamma\left[\frac{z_1 - \mu_1}{\sigma_1}\right] + \beta$$




Residuals : Residual connection create a short paths for gradients to flow directly through the neural network ; gradient remains sufficiently large.
- Improve gradient flow.
- Enables training of Deeper Networks.

★ Feed Forward Neural Networks

It processes the output from the self - attention mechanism and helps in features transformation and learning complex representations.
- Adding non - linearity.
- Adds depth to the model. → more learning

FNN consists of :-
1. Tow fully connected layears.
   Expands the dimension first, then reduces it back.
2. Non - linearity
   Usually RELU is applied after first layer → non-linearity
3. Dropout (Regularization)
   Prevents overfitting.

✳ Decoder

The transformer decoder is responsible for generating the output sequence one token at a time, using the encoder's output and the previously generated tokens.

- Masked Multi-head attention
  1. I/p Embedding and positional embedding
  2. linear projection for Q, K, V
  3. Scaled Dot product attention
  4. Mask application
  5. Multi-head attention mechanism
  6. Concatination and final linear ~~program~~ projection
  7. Risidual connection and Layer normalizatio.

★ Masked Multi-head attention

The transformer decoder is autoregressive at [inference] time and non [autoregressive] at training time.

[inference] ↑ prediction
~~training~~

[autoregressive] ↑ next output depends on previous output (Time series analysis)

So the output of next word depends on previous word while prediction ~~be~~ so it is autogressive.
But while training if we use autoregressive model it will make the training slow and time consume instead so we parallel training.
While training we know, we have both input and output data so even if after one gives answer wrong it is corrected using force teaching using dataset.

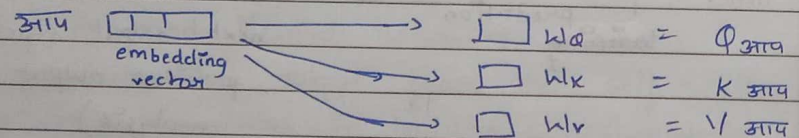
In parallel processing we calculate contextual vectors. So there's a problem

Example: English → Hindi
(translation)

How are you → आप कैसे हो।

(आप → 0.9 * V आप + 0.06 * V कैसे + 0.4 * हो

But at this point, this our first word prediction we don't know कैसे and हो is going to come we still ? have given the value →
this is a problem of data leakage.
In this type it will work good on training data but not on testing data.
So to solve the problem of data leakage and procces data parallely we use masked multi-head attention.




My for कैसे and हो.

| Q आप |
| --- |
| Q कैसे |
| Q हो |

Query matrix

| V आप |
| --- |
| V कैसे |
| V हो |

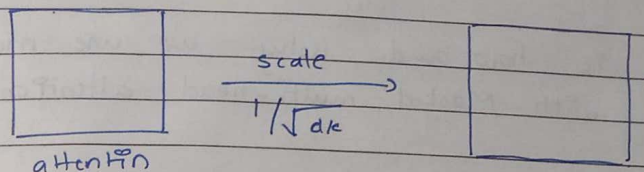
Value matrix

| K आप |
| --- |
| K कैसे |
| K हो |

Key matrix

Then we calculate attention score.

Q * k = atten score.




then we apply softmax to the score using which we calculate contextual embedding.

softmax →

| $w_{11}$ | $w_{12}$ | $w_{13}$ |
| --- | --- | --- |
| $w_{21}$ | $w_{22}$ | $w_{23}$ |
| $w_{31}$ | $w_{32}$ | $w_{33}$ |

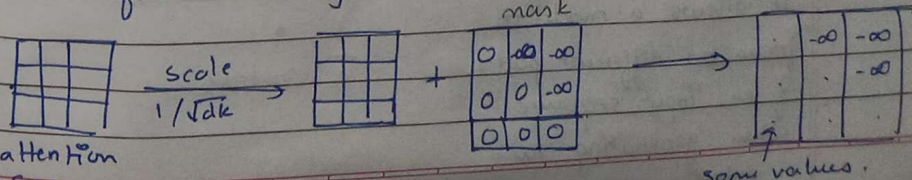
Contextual vector :-

आप = $w_{11}$ x V आप + $w_{12}$ * V कैसे + $w_{13}$ * V हो

कैसे = My for कैसे and हो।

So while calculating आप we do not need कैसे ka contribution and हो ka contribution
My for कैसे we do not need contribution of हो.

So for that we need $w_{12} = w_{13} = w_{23} = 0$
if the weights are zero then the term will be zero and thus no contribution.
So to make it happen we use masked modul.
So after scaling the matrix



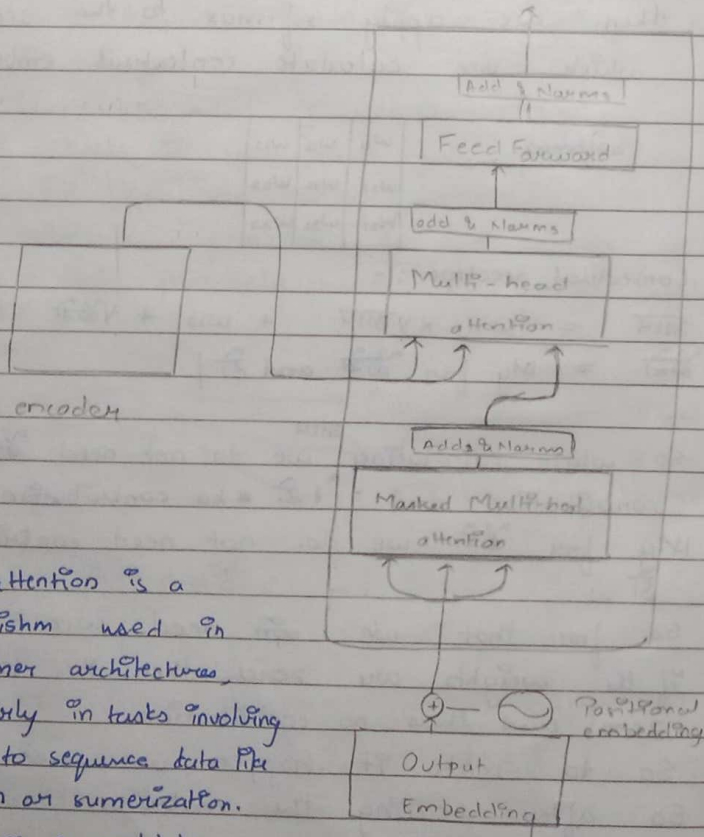
softmax(-∞) = 0

softmax →

| | | |
|---|---|---|
| $w_{11}$ | 0 | 0 |
| $w_{21}$ | $w_{22}$ | 0 |
| $w_{31}$ | $w_{32}$ | $w_{33}$ |

This is how and why we use mask. Done with Masked multi-head attention.

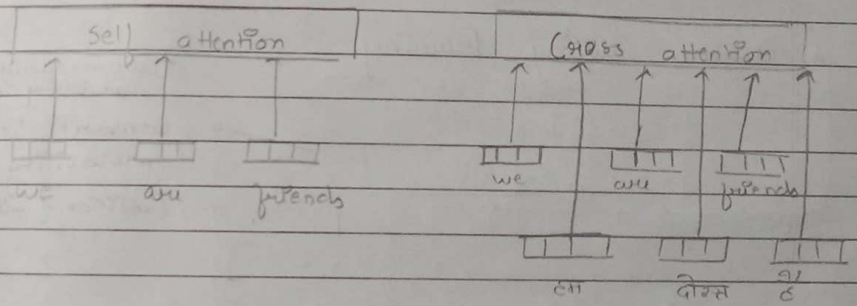
★ Multi-Head attention. (Decoder) also known as cross attention.




Cross-attention is a mechanism used in transformer architectures, particularly in tasks involving sequence to sequence data like translation or sumerization. It allows a model to focus on different parts of an input sequence when generating an output sequence.

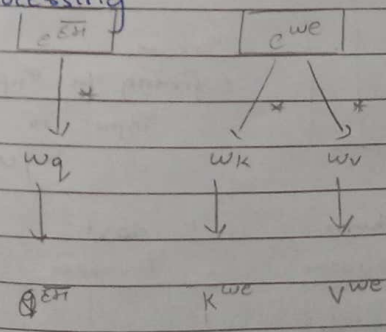
The decoder generates output one word at a time, depending on:
- What was been generated so far.
- The information recieved from encoder (input sequence).

Cross attention is conceptually similar to self attention but differs in:
- Input




Self attention use a single sequence (input)
Cross attention use two sequence (input and ouput)

★ Processing



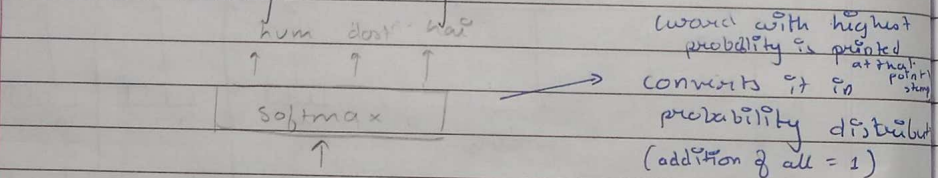

Why for Q, K, V for दोस्त and है?

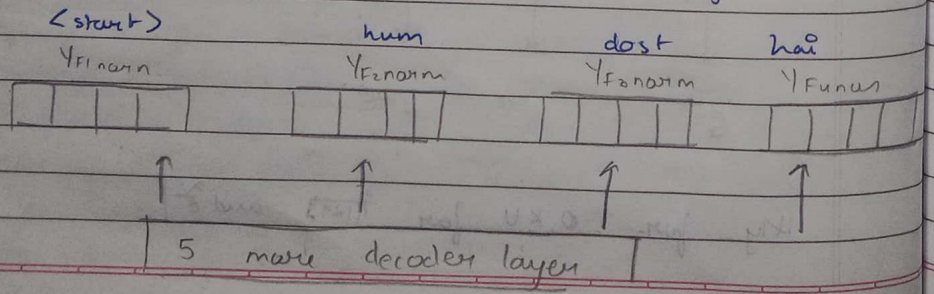
- Generates query vector from the output sequence (हम है) and key/value vector from the input sequence.
Performing dot products to derive attention scores, which indicates the relationship between input sequence and output sequence

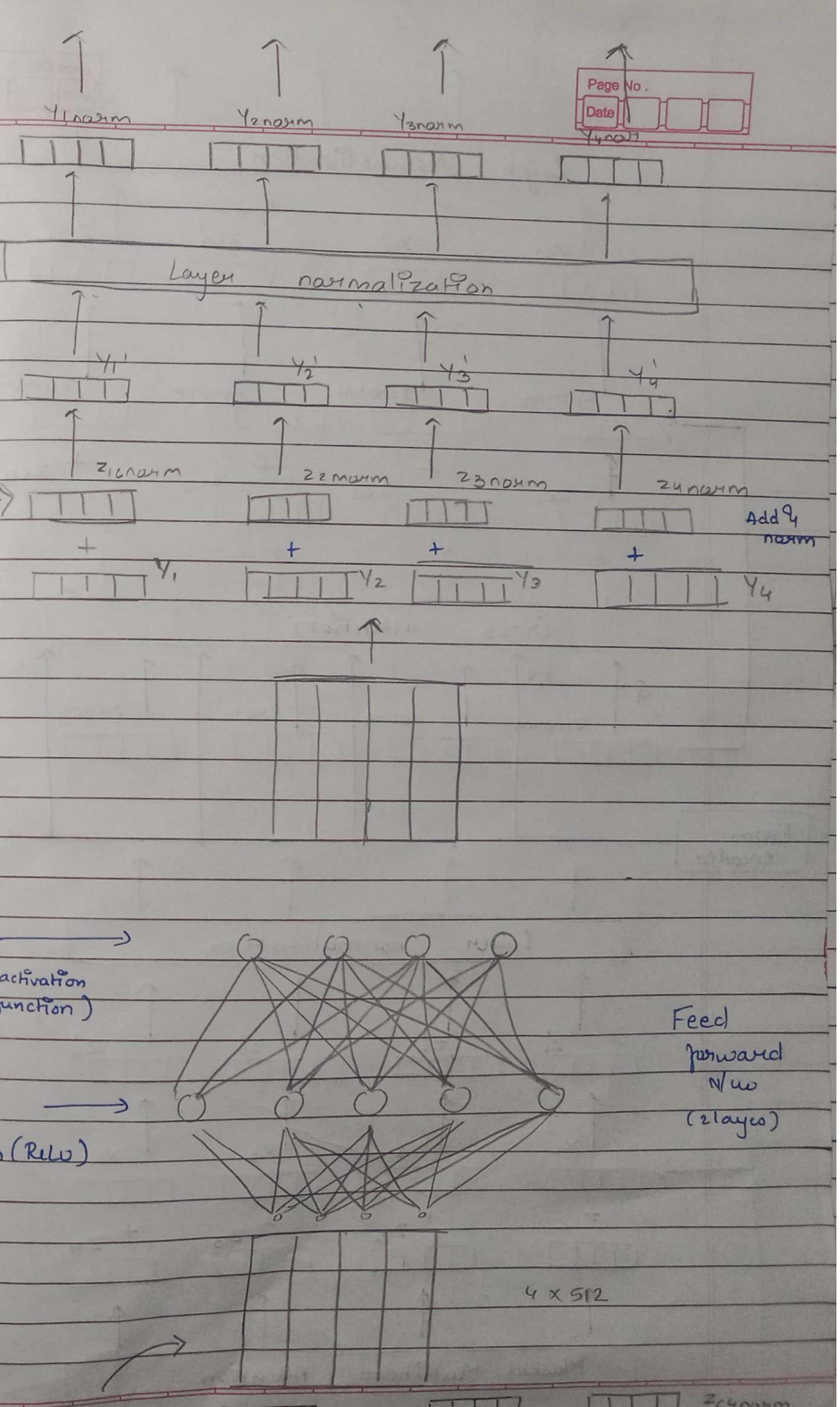
we are friends

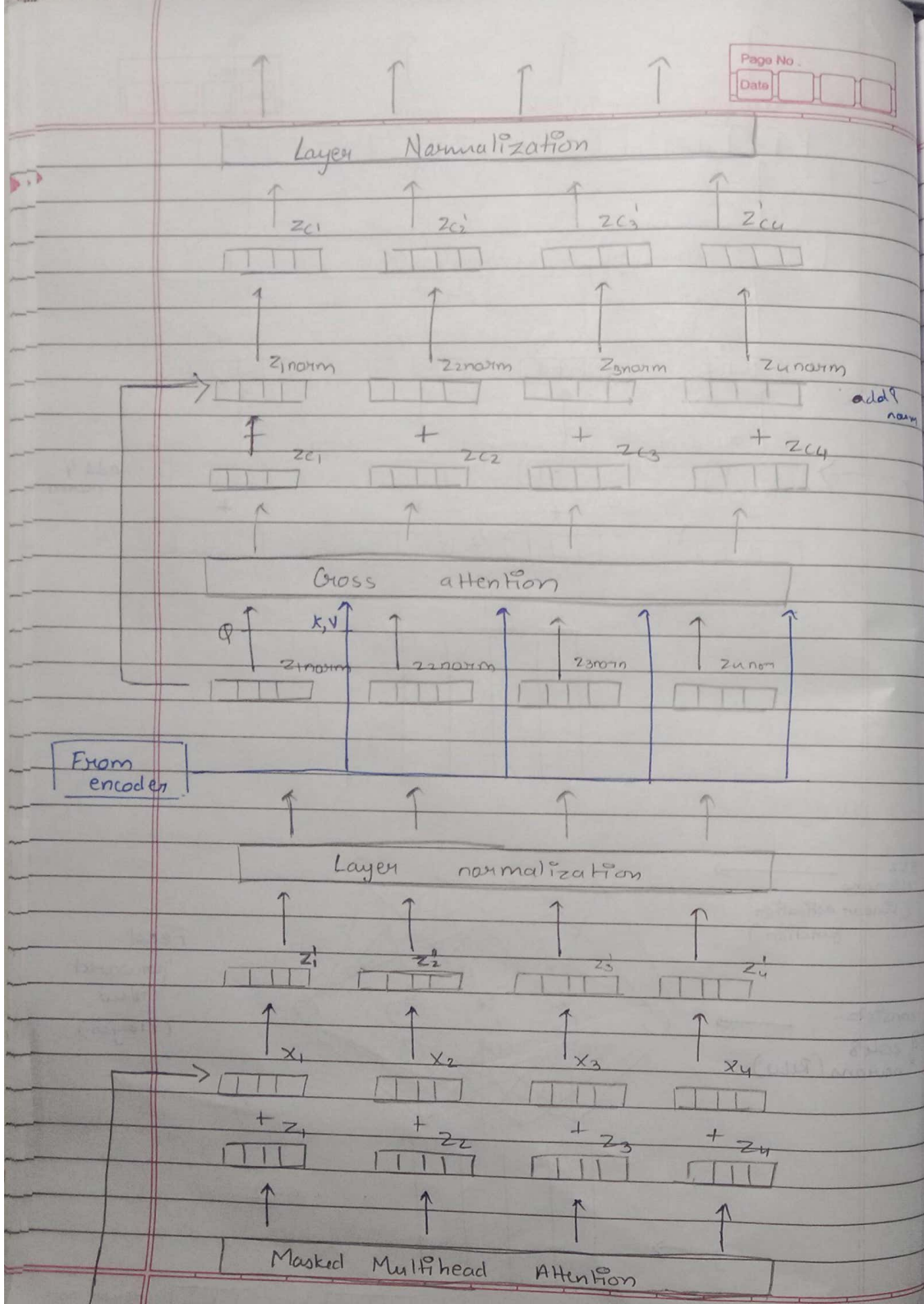
| | | | |
|---|---|---|---|
| हम | ○ | . | . |
| दोस्त | . | . | ○ |
| हैं | . | ● | . |

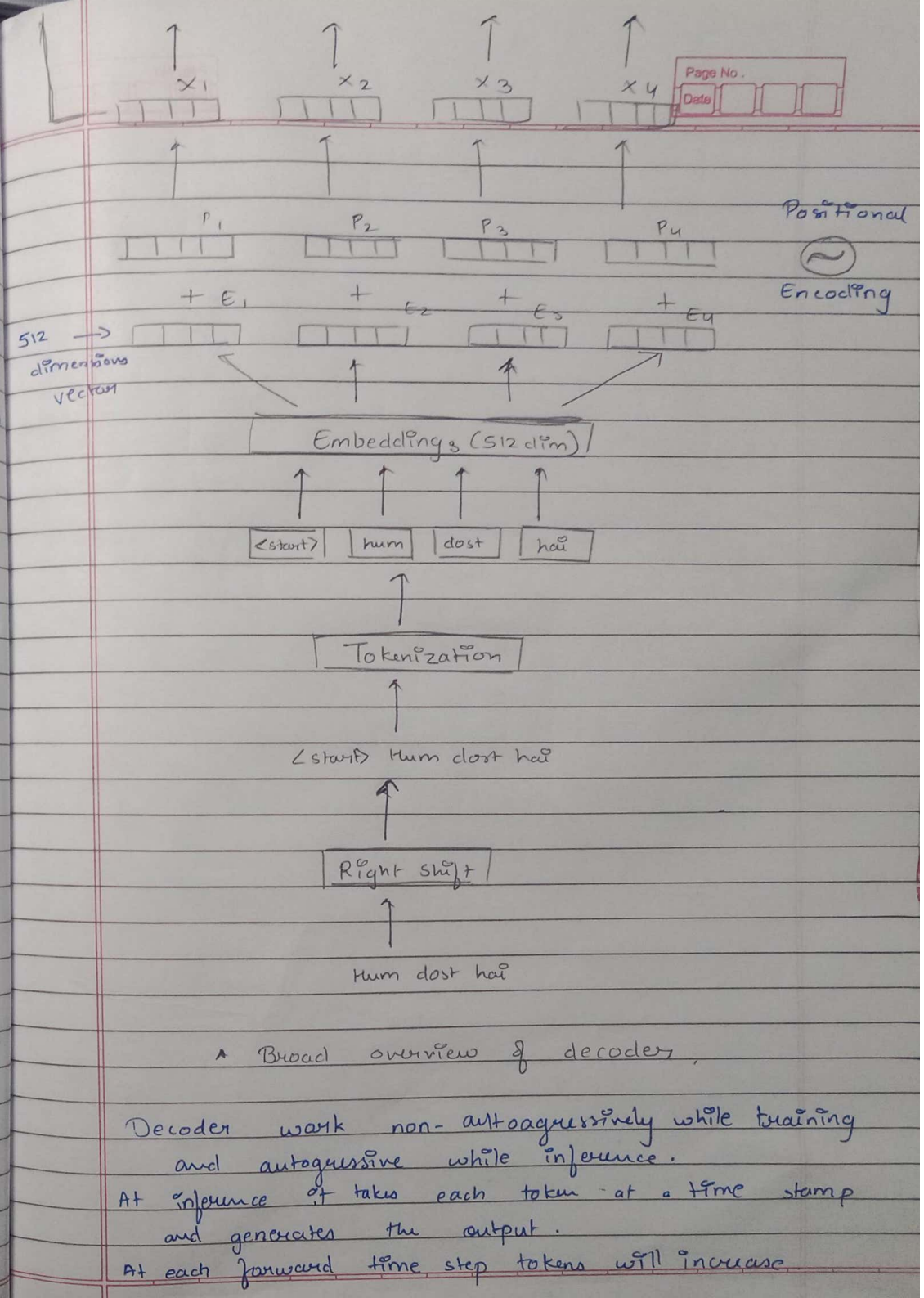
★ Decoder during training






* Broad overview of decoder.

Decoder work non-autoagressively while training and autogressive while inference.

At inference it takes each token at a time stamp and generates the output.

At each forward time step tokens will increase.