Sequence to Sequence Models:-

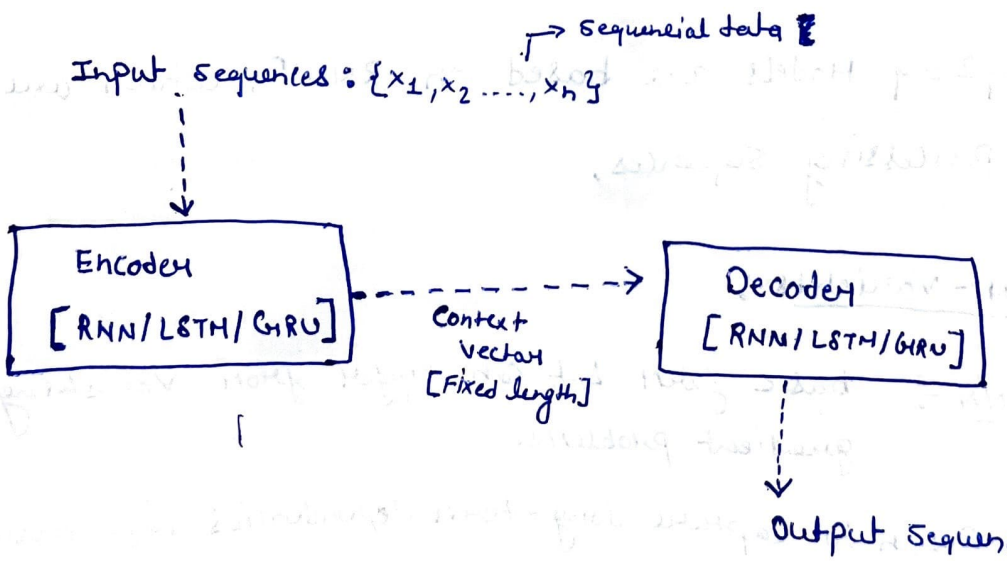
→ Sequence-to-Sequence (Seq2Seq) Model are a class of Models design to handle tasks where Input and Output are sequences of varying lengths.

→ They are widely used In natural language processing (NLP) tasks such as:- {Applications}

1. Machine translation:- Covert Text from one language to another.
2. Text summerization:
3. dialouge systems:- Buiding Conversational agents.
4. Text-generation:- Generating human-like text based on a given promt. OR chatbot.
5. Speech-to-text.

Basic structure {Seq2Seq}

→ Encoder-Decoder Architecture, the first Seq2Seq Model.



* NOTE:- It is Not Nessesary that the length of the Input & output seq is same

Encoder - Decoder Architecture.

Encoder:

↳ Process the input sequences and encodes it into a fixed-length context vector (thought vector). This context vector captures the essence of the input sequences.

Decoder:

↳ Takes the context vector from the encoder and generates the output sequences.

The output is produced token by token, usually using the previously generated token as input for the next step.

RNN-Based Seq2Seq Models:

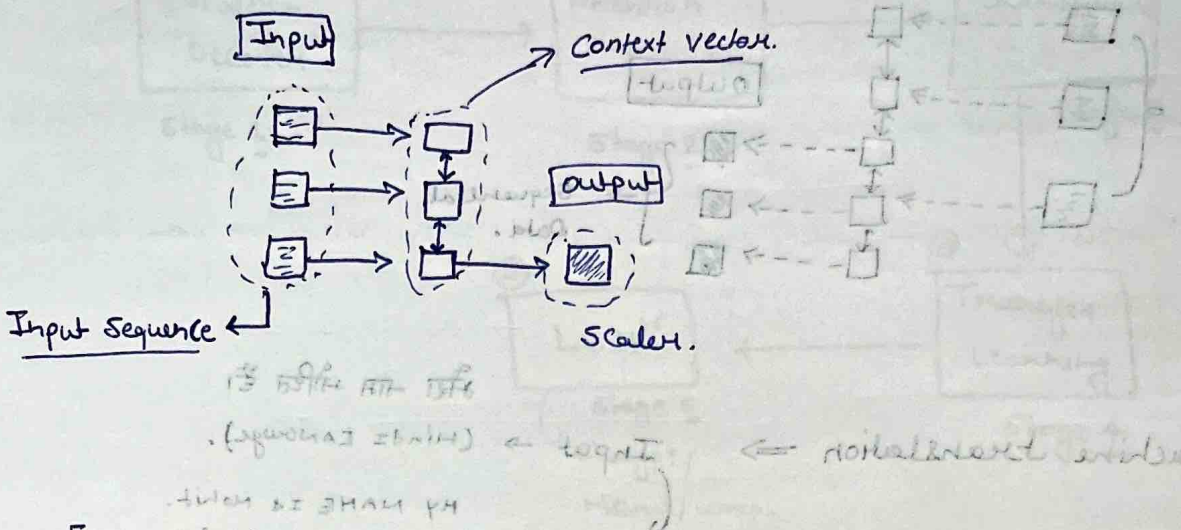
- Tradition seq2seq models are based on RNN's, which are well-suited for processing sequences.

- Common RNN-variants:

1. Simple-RNN: basic form but can suffer from vanishing gradient problems.
2. LSTM: Design to capture long-term dependencies and avoid vanishing gradients.
3. GRU: A simpler alternatives to LSTM with similar performance.

Types of RNN :-

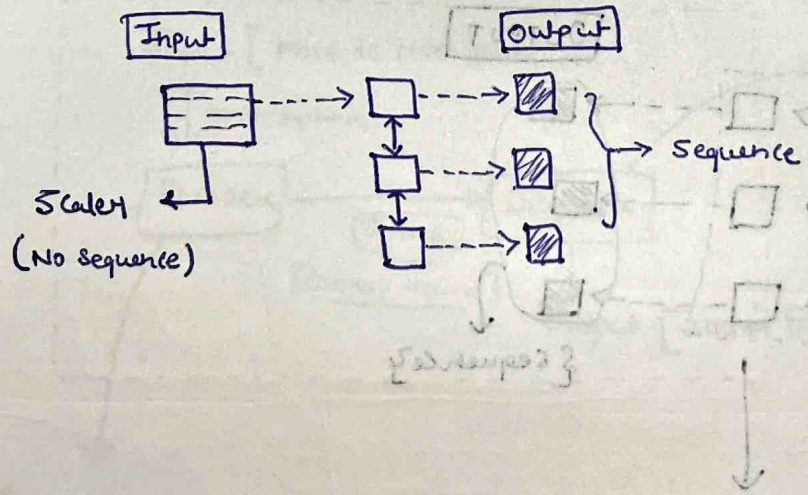
①. Many to ONE :-



Ex→ Sentimental Analysis ⇒ Input → Movie Rewies.

↳ Output → predict (positive, Negative, Neutral).

②. ONE to MANY :-



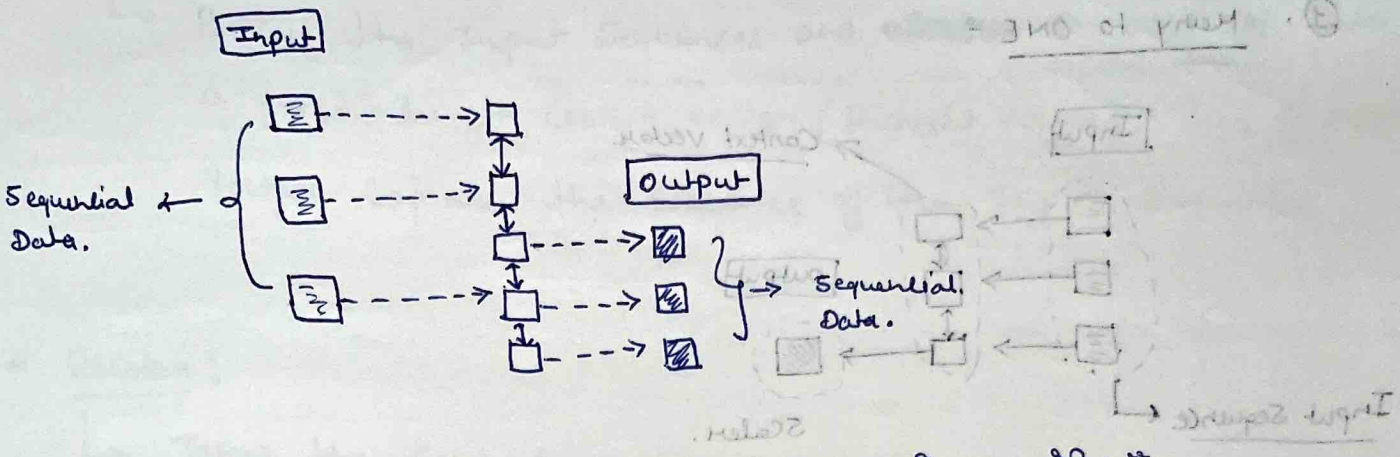
Ex→ Image Captioning ⇒ Input → Image (Image of something)

↳ Output → Discription → "Man IS Riding Hourse".

Imp.

③. MANY to MANY :- {ASYNCHRONUS}

↳ Having Input & diag size, output & diag.



मेरा नाम मोहित है।

Ex→ Machine translation ⟹ Input → (Hindi Language).

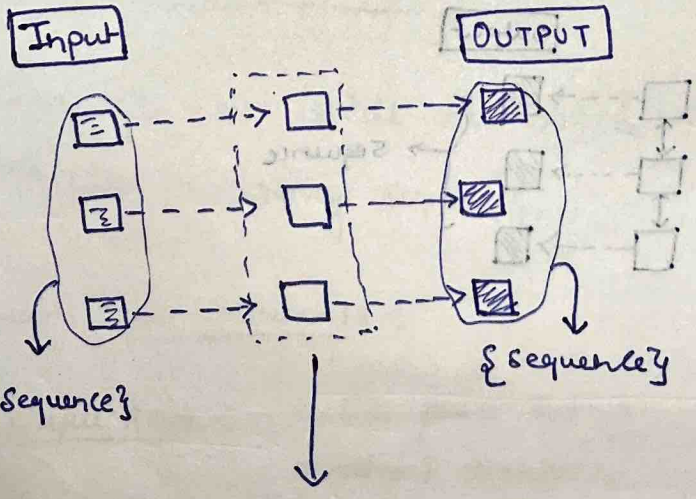
MY NAME IS MOHIT.

Output → (English Language).

NOTE → Input MAY have different sequence and output may have different sequence.

→ Sequence to Sequence Model designed to solve this Many to Many Problem.

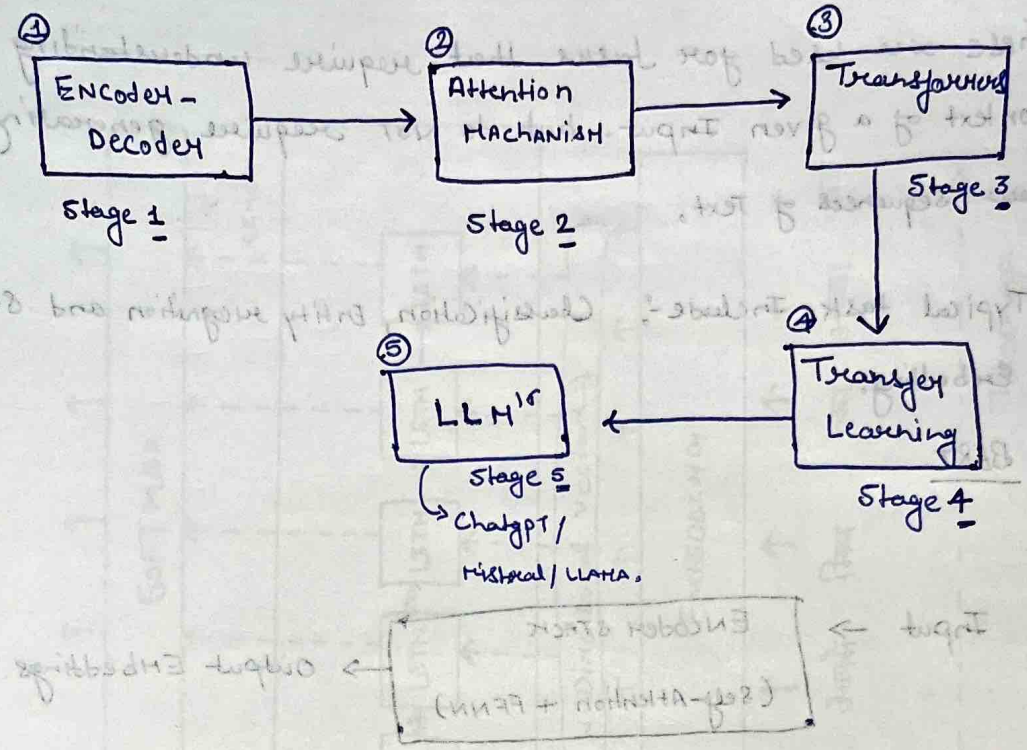
④. MANY to MANY :- {SYNCHRONUS}



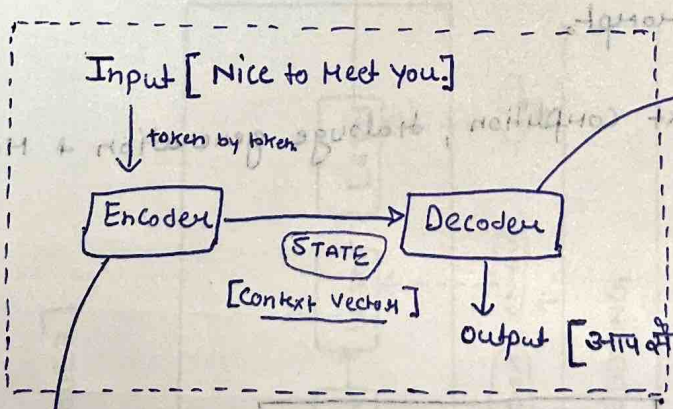
USING :- ①. Parts of speech tagging.

②. Name Entity recognition.

History of Seq to Seq Models :-



STAge-1 : Encoder - Decoder Architecture



Decoder → It takes the given compressed Information. and generates output one by one (or word by word).

Encoder → It process the Input word by word. and compress the all Information.

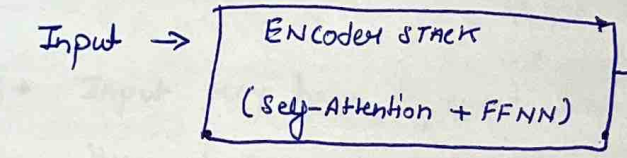
NOTE→

① Encoder-only Models :-

↳ These are used for tasks that require understanding the context of a given Input. but do NOT require generating new sequences of Text.

→ Typical task Include :- Classification, Entity recognition and Sentence Embedding.

→ BERT.

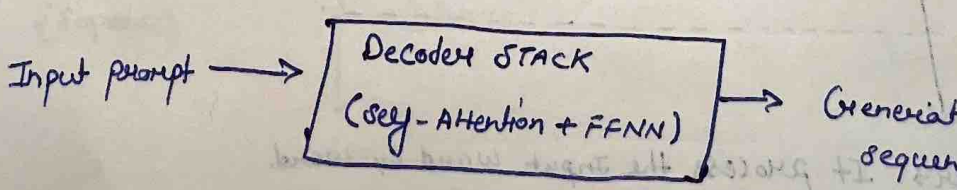


② Decoder-only Models :-

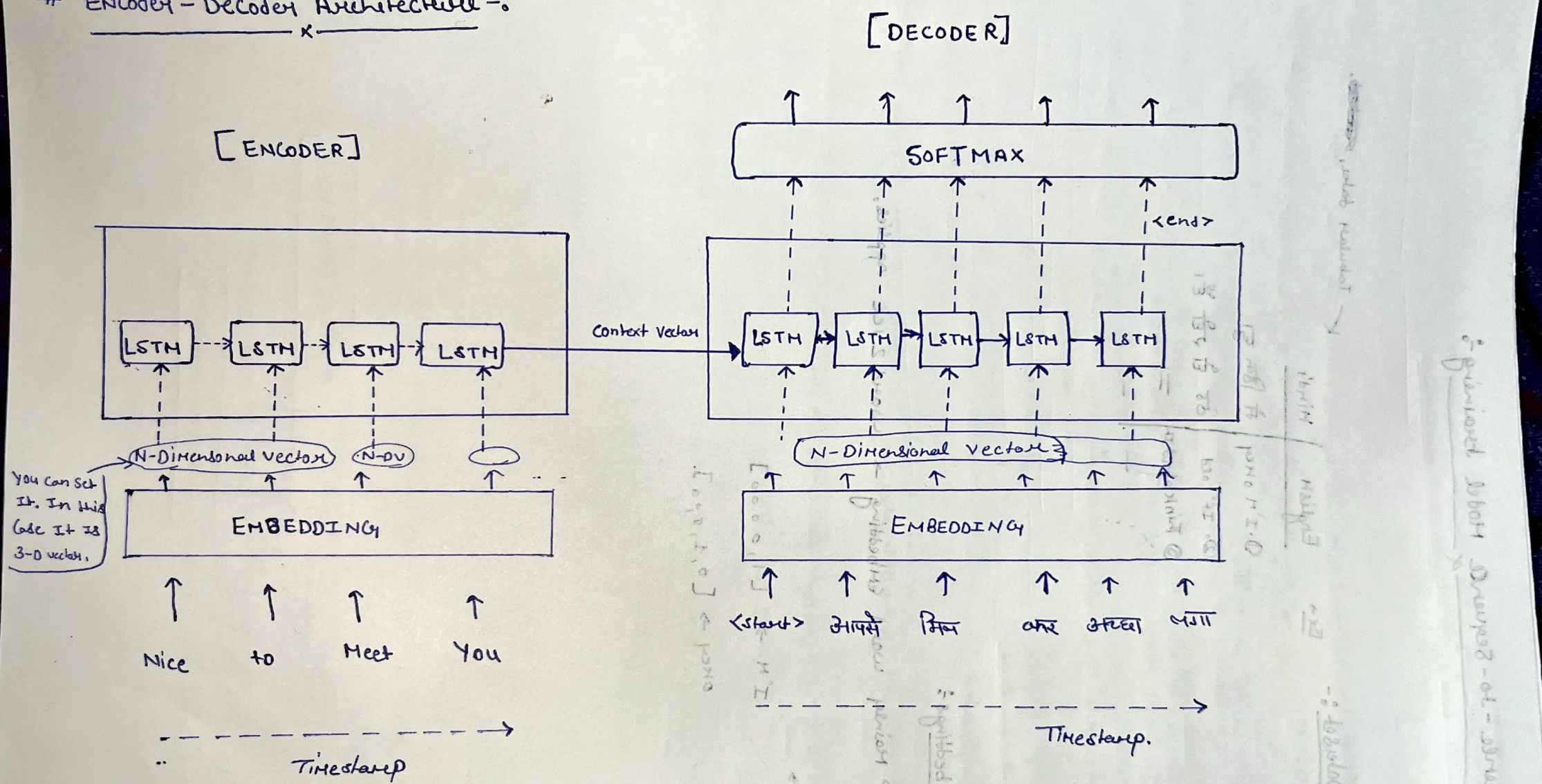
↳ They are used for generating Contextualy text based output on a given prompt.

→ USED In :- text completion, dialouge generation & More.

→ GPT



Encoder - Decoder Architecture :-



STAGE-2 :- ATTENTION MACHANISM.

↳ Attention Machanism Is a technique In ML & NLP that allows Models to focus on different parts of the Input sequence when producing an output.

→ It helps Model weigh the Importance of different Input elements dynamically, rather than treating all Inputs elements Equally.

Key Components :-

Ⓐ. Query :- Represents the Item for which attention Is being computed.

Ⓑ. Key :- Represents the Items In the Input sequences to which the Model Is attending.

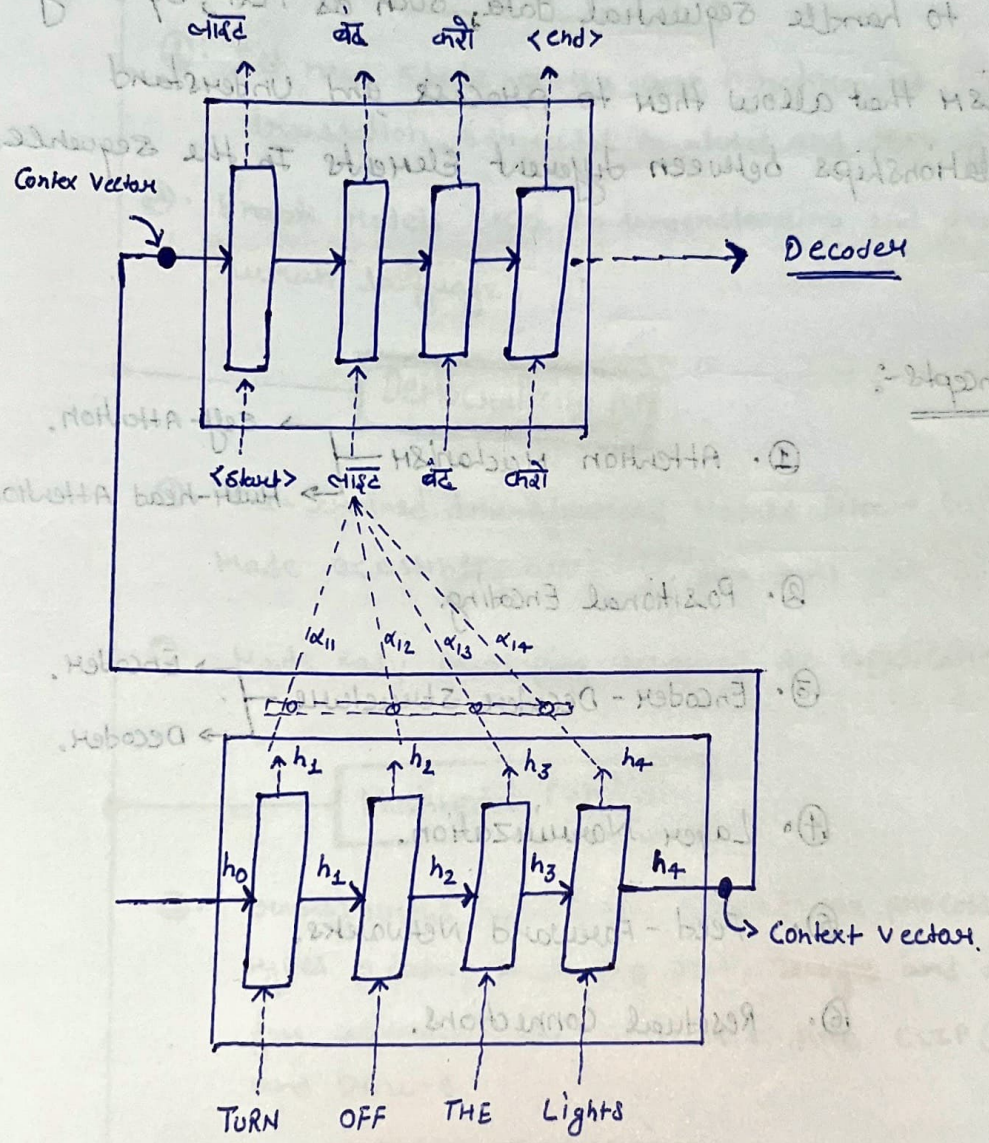
Ⓒ. Value :- Represents the actual Information or Content of the Input sequences.

Formulas :-

Ⓐ. Attention Score Calculation ⇒ $\text{Score}(Q, K) = Q \cdot K^T$

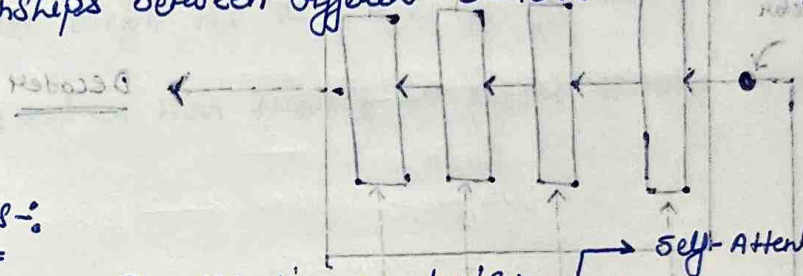
Ⓑ. Attention Weights ⇒ $\text{Weights} = \text{SoftMax}(\text{Score}(Q, K))$

Ⓒ. Context Vector ⇒ $\text{Context} = \text{Weights} \cdot V$


लाइट
बंद
करो
<end>
Context Vector
Decoder
<start>
लाइट
बंद
करो
α_{11}
α_{12}
α_{13}
α_{14}
h_1
h_2
h_3
h_4
h_0
h_1
h_2
h_3
h_4
Context Vector.
TURN
OFF
THE
Lights

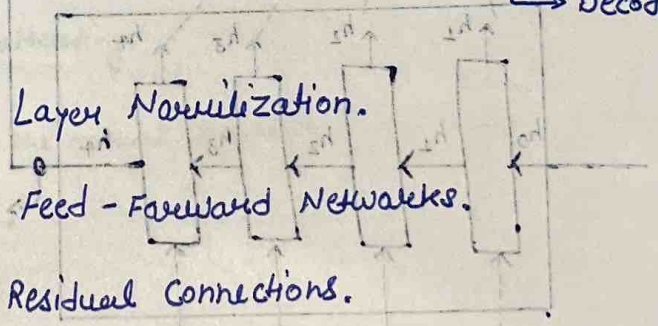
* STAGE-3 :- Transformer

↳ Transformers are a type of neural network architecture designed to handle sequential data, such as text, by using mechanism that allow them to process and understand the relationships between different elements in the sequence.



Key Concepts :-

1. Attention Mechanism
   - Self-Attention.
   - Multi-head Attention.
2. Positional Encoding.
3. Encoder - Decoder structure
   - Encoder.
   - Decoder.
4. Layer Normalization.
5. Feed - Forward Networks.
6. Residual Connections.



Applications :- NLP, Vision, Speech.

1. Advantages ⇒ Parallelization & Scalability.

2. Disadvantages ⇒ Computationally Expensive & Data Hungry.

Impact of Transformers :-

Revolution In NLP

1. Set new state-of-the-art benchmarks in tasks like :- translation, sentimental Analysis and text-generation.
2. Enable Models excel in understanding and generating human language.

Democratizing AI

1. Pre-trained transformers Models like → GPT and BERT are Made acessible, allowing fine-tune for specific tasks.
2. Made easy developing advanced AI applications.

Multimodal Capability

1. Transformers have been adapted for processing multiple types of data, including text, Images and audio, leading for advancements in Models like CLIP (for tex-to-Image) and DALL-E.

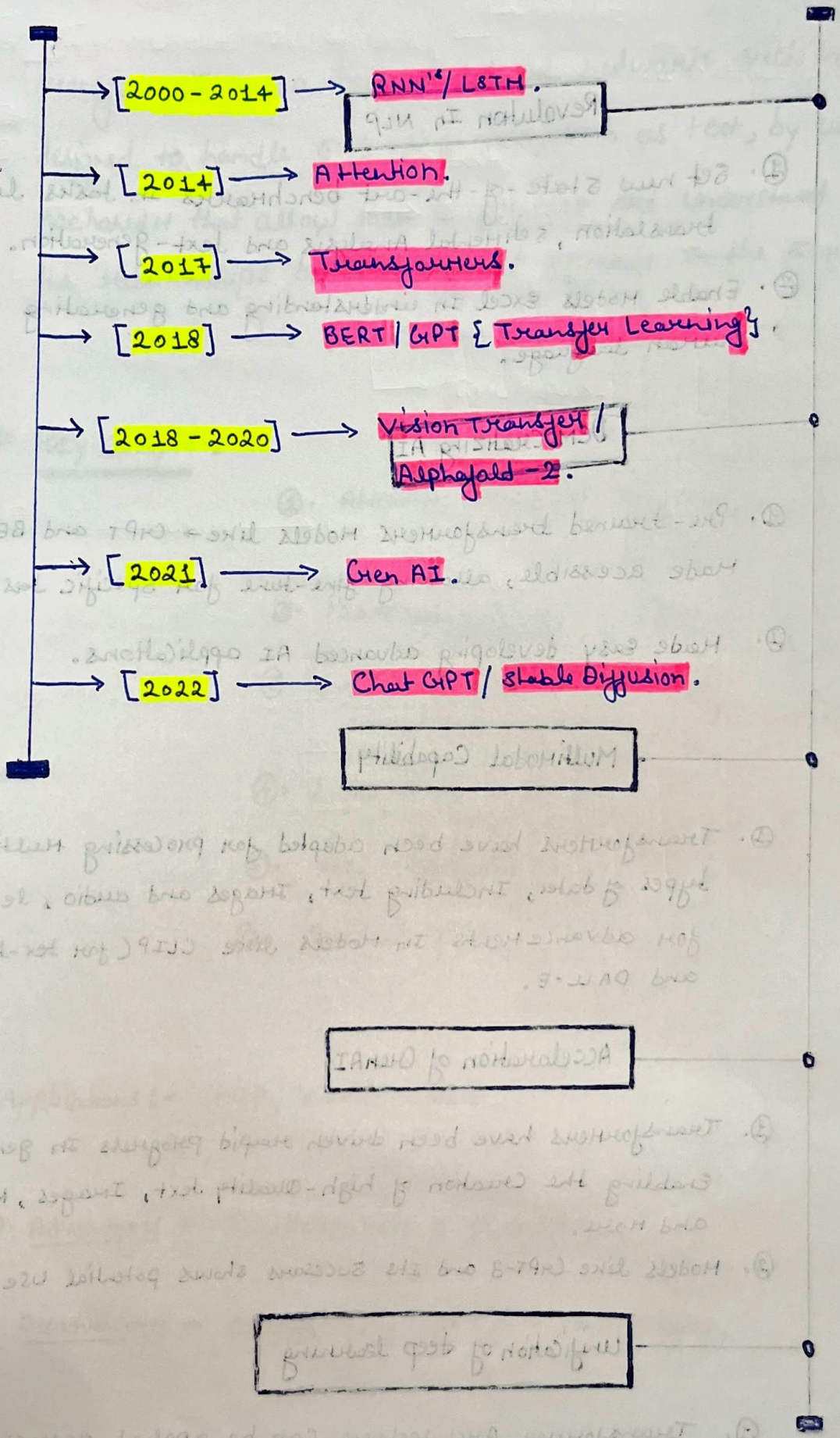
Acceleration of GenAI

1. Transformers have been driven rapid progress in gen AI, Enabling the Creation of high-Quality text, Images, Music and More.
2. Models like GPT-3 and its sucesors shows potential use of LLM's.

Unification of deep learning

1. Transformer Architecture can be applied across various domains, from NLP to vision, and even to scientific data, Pushing towards a More unified deep learning framework.

The Timeline :



Advantages :

1. Scalability.
2. Transfer Learning.
3. Multimodal.
4. flexible Architecture
   - Encoder only (BERT).
   - Decoder only (GPT).
5. Ecosystem
   - huggingface.
   - Video / Blogs.
6. Integrated AI Media.

Disadvantages :

1. High Computation (GPU).
2. Large Data & text → unsupervised labeling}
3. overfitting.
4. Energy Consuption.
5. Interpreter.
6. Bias → Ecthical Conserves.

Future :-

1. Improvements in efficiency.
   - Pruning.
   - Quantization.
   - Knowledge Distillation.

2. Multimodal Capabilities
   - text.
   - Image / speech.
   - Sensory data.
   - Biometric feedback.
   - Time Series.

3. Responsibility
   - Bias.
   - Ethical Concerns.

4. Domain Specific
   - Doctor GPT.
   - Teacher GPT.
   - Legal GPT.

5. Multi-lingual → English, Hindi, French. etc.

6. Interpretability → Critical Domains like finance, Medical etc.

Till now Transformers are Black-box Model.

① Attention Mechanism

Ⓐ. Self-Attention :-

↳ Self-Attention is a mechanism used in neural networks, especially in models like transformers to help the network focus on different parts of the input data more effectively.

Simple Explanation :- Imagine you're reading a sentance, and you want to understand the meaning of a word by looking at the surrounding words.

Self-Attention allows the model to look at each word ex- "cat", the model might pay attention to "sat" and "on" because they are more relevant to understanding the context around "cat".

Steps In Self-Attention :-

1. **Input Words** :- Let say we have a sentance with words as input.

2. **Create Attention Scores** :- For each word, the model calculates a score (or weights) that shows how much attention to give to every other word.

3. **Weighted sum** :- The model then multiplies each word's embedding by its corresponding attention score and sums them up to get a new representation of the word.

Workflow :-

Step 1 :- **Input Words** ⇒ A sentence like "The Cat Sat" is input the Model.

Step 2 :- **Attention Scores** ⇒ A Table shows the attention scores for each pair of words.

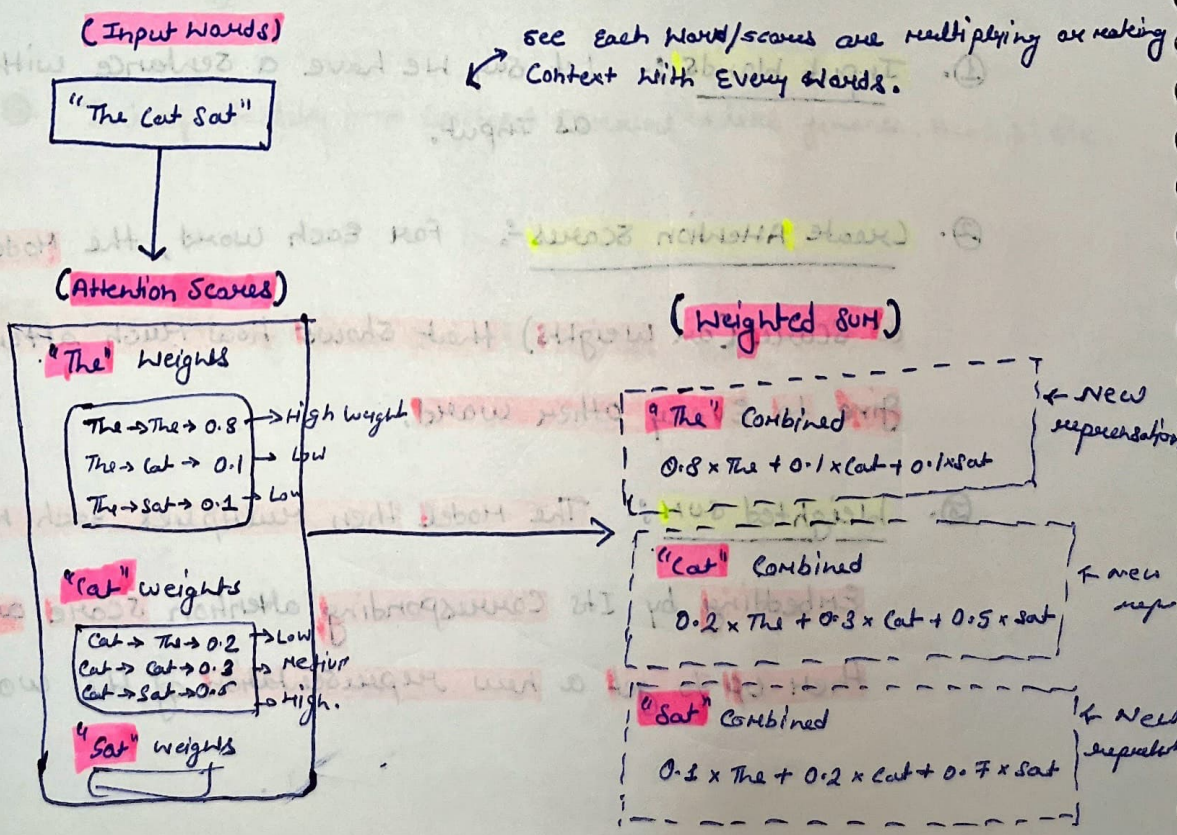
Ex →

Ⓐ. "The" pays most attention to "The" and some to "Cat" and "Sat".

Ⓑ. "Cat" pays more attention to "Sat" then "The".

Step 3 :- **Weighted sum** ⇒ These **scores** are used to **create a new representation of each word**, which takes into account the context provided by other words.

Diagram :-



First principle Approach

Money BANK Grows

River bank flows

* Notice both sentence have "bank" word with different meaning.

bank → 0.3 Money + 0.7 bank + 0.1 grows

bank → 0.5 river + 0.4 bank + 0.1 flows

* See both embedding of bank now different because of Contexual Embeddings.

Ex→

Money = 0.7 Money + 0.2 bank + 0.1 grows

bank = 0.25 Money + 0.7 bank + 0.05 grows

grows = 0.1 Money + 0.2 bank + 0.7 grows

Word Embedding (S_1)

river = 0.8 river + 0.15 bank + 0.05 flows

bank = 0.2 river + 0.78 bank + 0.02 flows

flows = 0.4 river + 0.01 bank + 0.59 flows

Word Embedding (S_2)

n-dimension (here 2 words so (3×3) → 3-Dimension)

(new)

$e_{Money} = 0.7\, e_{Money} + 0.2\, e_{bank} + 0.1\, e_{grows}$

$e_{bank} = 0.25\, e_{Money} + 0.7\, e_{bank} + 0.05\, e_{grows}$

$e_{grows} = 0.1\, e_{Money} + 0.2\, e_{bank} + 0.7\, e_{grows}$

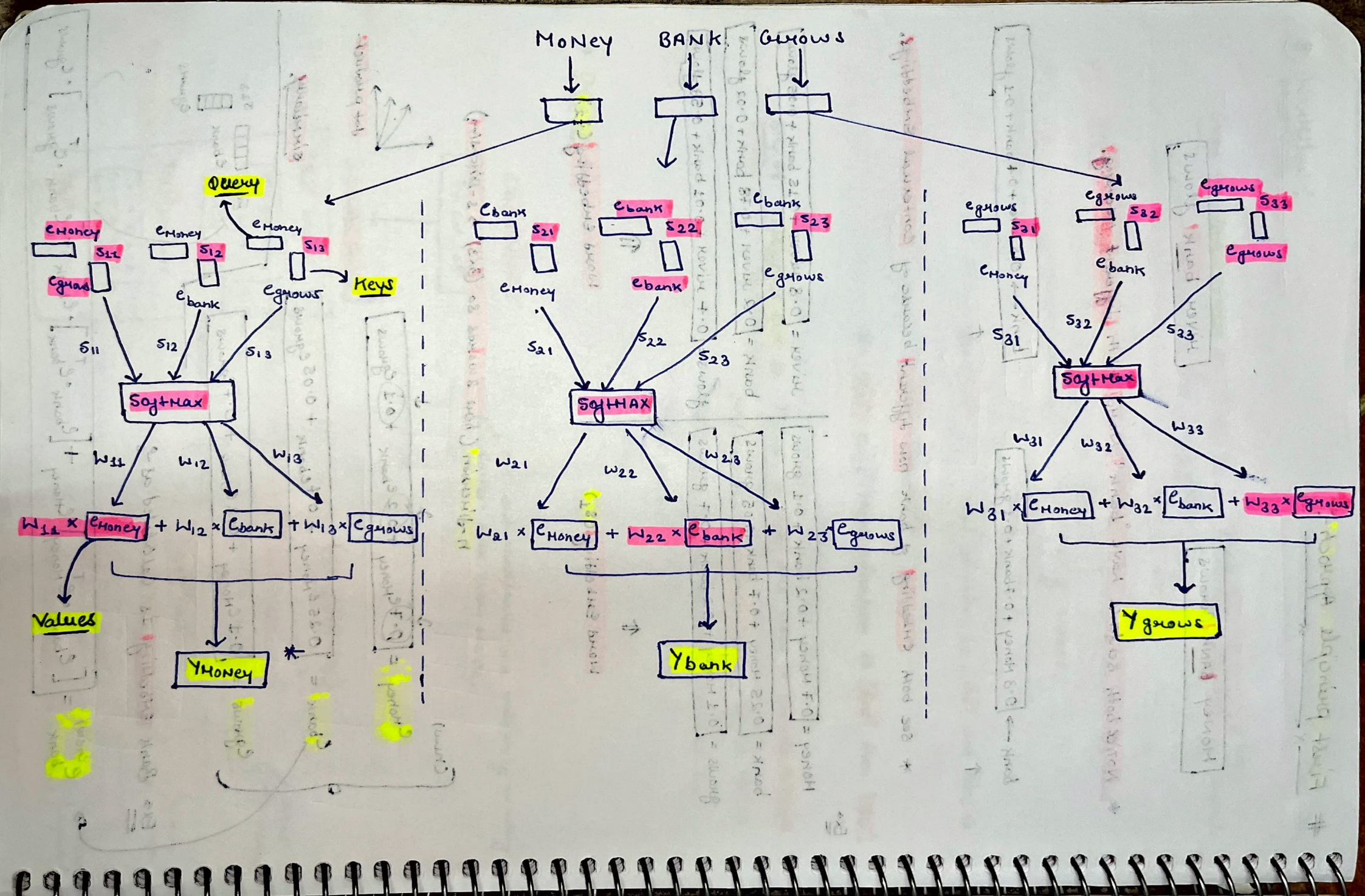
dot product.

similarly.

Ex→ e_{bank} S_{23} e_{grows}

Ex→ Bank Embedding is Calculated as →

$$e^{(new)}_{bank} = \left[e_{bank} \cdot e^{T}_{Money}\right] \cdot e_{Money} + \left[e_{bank} \cdot e^{T}_{bank}\right] \cdot e_{bank} + \left[e_{bank} \cdot e^{T}_{grows}\right] \cdot e_{grows}$$


MONEY
BANK
GROWS
Query
Keys
Values
Soft+Max
Soft+MAX
Soft+Max
YMONEY
Ybank
Ygrows

Key operations :-

↳ The main steps in the self-Attention Mechanism are :-

① . "Query", "Key" & "Value" Matrices :- Each token's Embedding is linearly transformed into three vectors :-

(A) . Query Vector (Q) :-

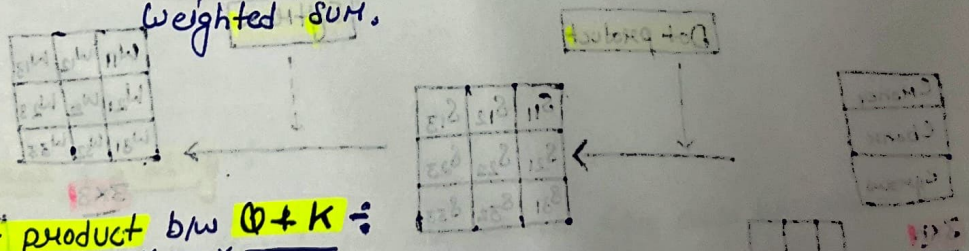
↳ Represents the current token being processed.

(B) . Key Vector (K) :-

↳ Represents other tokens to compare with.

(C) . Value Vector (V) :-

↳ Represents the actual token Embeddings used in the weighted sum.



② . Dot product b/w Q & K :-

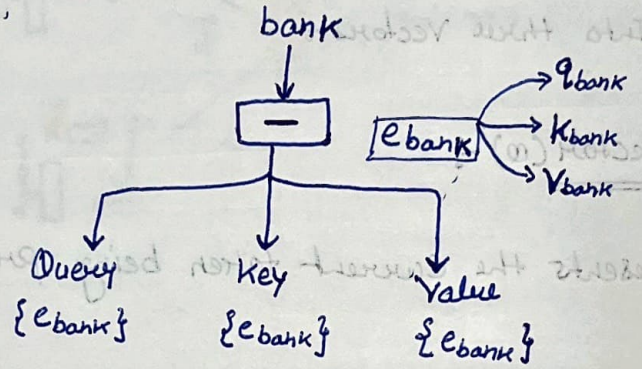
↳ The similarity between the query and key vectors is computed using a dot product.

③ . Softmax :- The resulting dot products are normalized using a softmax function to compute attention weights.

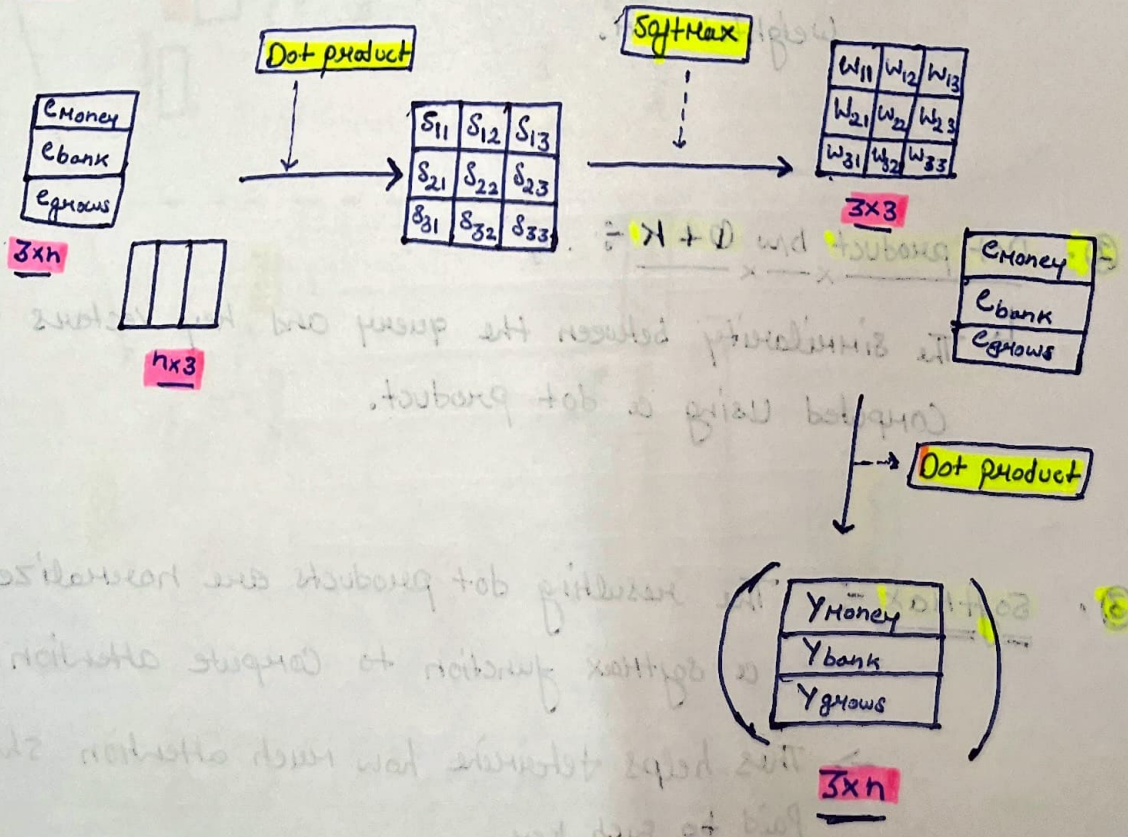
→ This helps determine how much attention should be paid to each key.

④. **Weighted sum of V :-**

↳ The attention weights are applied to the value vectors to get the final representation of each token.



Visualization :-



Parallelism :-

↳ Self-Attention have ability to process operations In parallel, making transformers well-suited for large-scale training on GPU's or TPU.

① Matrix Multiplications :-

↳ Instead of processing tokens one by one, The self-attention Mechanism can operate on the Entire Sequences of token In parallel.

→ This Is possible because Inputs (queries, keys and values) are Metrices and operations are dot-product.

② Attention Score Calculation :-

↳ Dot product are multiplying by Query Matrix (Q) with the transpose of the key Matrix (K^T).

$$\text{Attention Score} = Q \cdot K^T$$

③ Softmax + Scaling :-

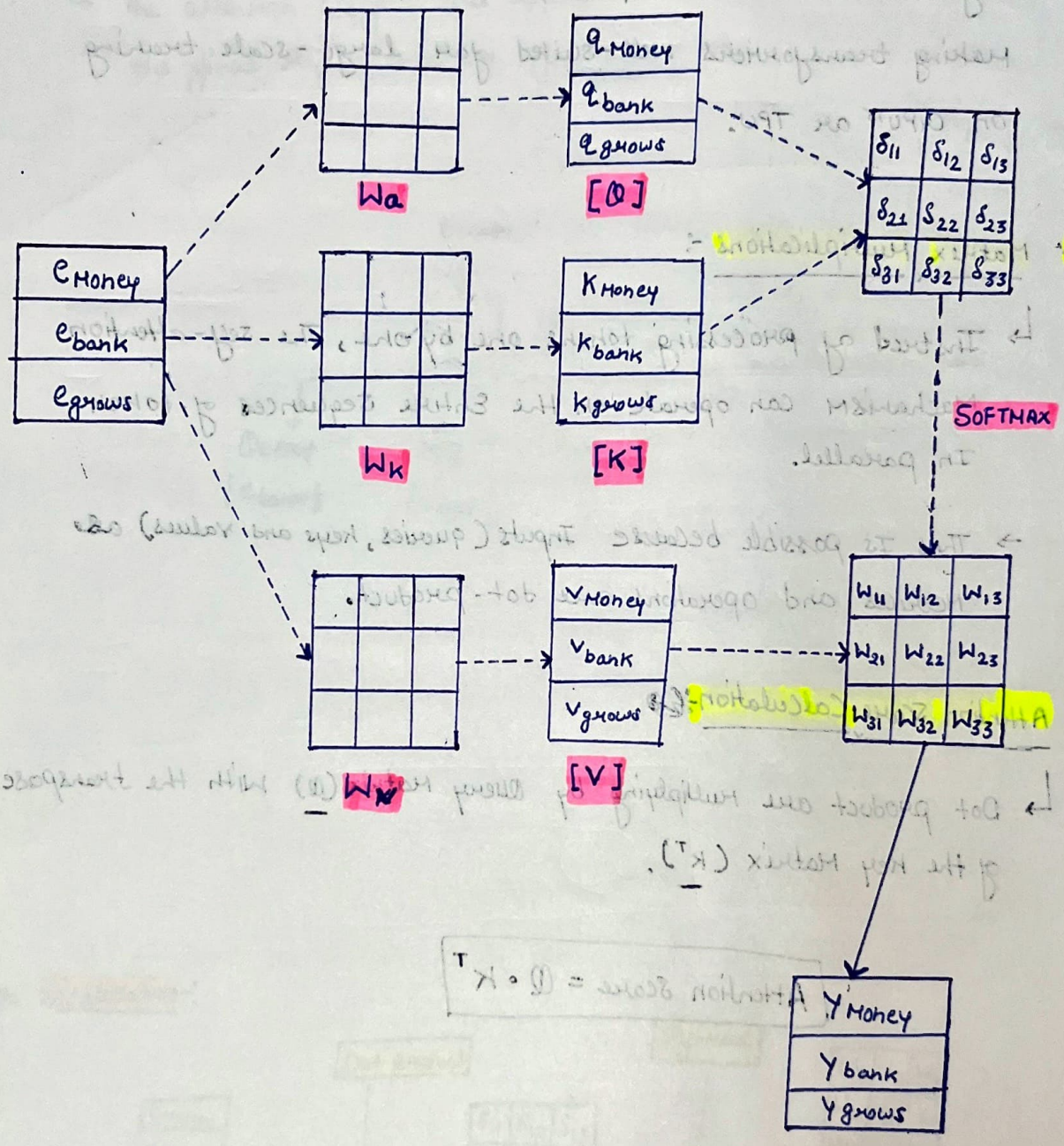
↳ The attention scores for all tokens can also be scaled and normalized In parallel.

$$\text{Normalized Weights} = \text{Softmax}[Q \cdot K^T]$$

④ Weighted SUM of Values :- The weighted SUM Is Calculated for all tokens In one go by Multiplying the Softmax output Matrix with the Value Matrix (V).

$$\text{Contex vector} = [(Q \cdot K^T) \cdot V]$$

Complete Self-Attention Structure :-



Scaled Dot Product Attention :

↳ Scaled Dot-production Attention is a key mechanism used in transformer models, especially in NLP tasks like, machine translation, text generation and summerization.

Flow :

1. Money → e_{Money} → Money Bank Grows {Embeddings}

e_{Money} → W_Q, W_K, W_V

W_Q → q_{Money}, W_K → K_{Money}, W_V → V_{Money}

W_Q, W_K + W_V are parameter Matrices.
The data/weights in these matrices are created by Embedding itself.
And using backpropogation these weights updated till becomes accurate.

↳ Query, key + Value vectors.

→ these vectors stack-up together.

2. Q Matrix: q_{Money}, q_{bank}, q_{grows} — K Matrix: K_{Money}, K_{bank}, K_{grows} — V Matrix: V_{Money}, V_{bank}, V_{grows}

3. Q (3×h) dot K (h×3) → S_{11} S_{12} S_{13} / S_{21} S_{22} S_{23} / S_{31} S_{32} S_{33} (Scaling *) → Softmax → W_{11} W_{12} W_{13} / W_{21} W_{22} W_{23} / W_{31} W_{32} W_{33} (3×3) → Y_{Money}, Y_{bank}, Y_{grows}

3×h

$$Attention(Q, K, V) = Softmax(Q \cdot K^T) \cdot V$$

Scaled Dot-Product :- Formula.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V$$

Scaling ($\sqrt{d_k}$) :-

↳ The dot product values are divided by the square root of the dimensionality of the keys, $\sqrt{d_k}$ to avoid overly large values.

→ Without the scaling, large values in the dot product could lead to gradients becoming very small or exploding, making training difficult.

$$\text{Scaled score} = \frac{Q \cdot K^T}{\sqrt{d_k}}$$

NOTE → for a 3-D vector, the d_k will be → 3

Key points :-

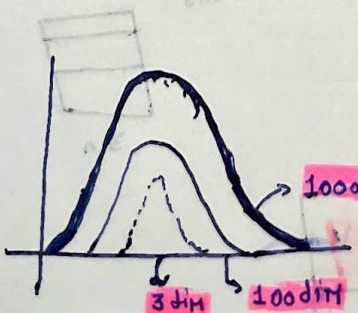
↳ low dimensional → dot product → low variance

→ high dimensional → dot product → high variance.

Graph →

⇒ high variance

It's a problem.

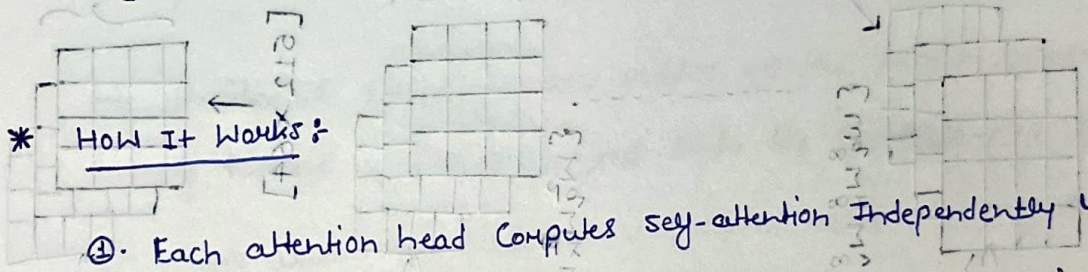


See, low dimension have lower variance and vice-versa.

Multi-Head Attention :-

↳ Multi-head attention is a key mechanism in the Transformer architecture that allows the model to capture different types of relationships between tokens in a sequence.

→ Instead of using a single attention function, multi-head attention applies multiple attention mechanisms (or heads) in parallel, allowing the model to focus on various parts of the sequence simultaneously.



* How it works :-

① Each attention head computes self-attention independently using different learned linear projections of the original queries, keys and values.

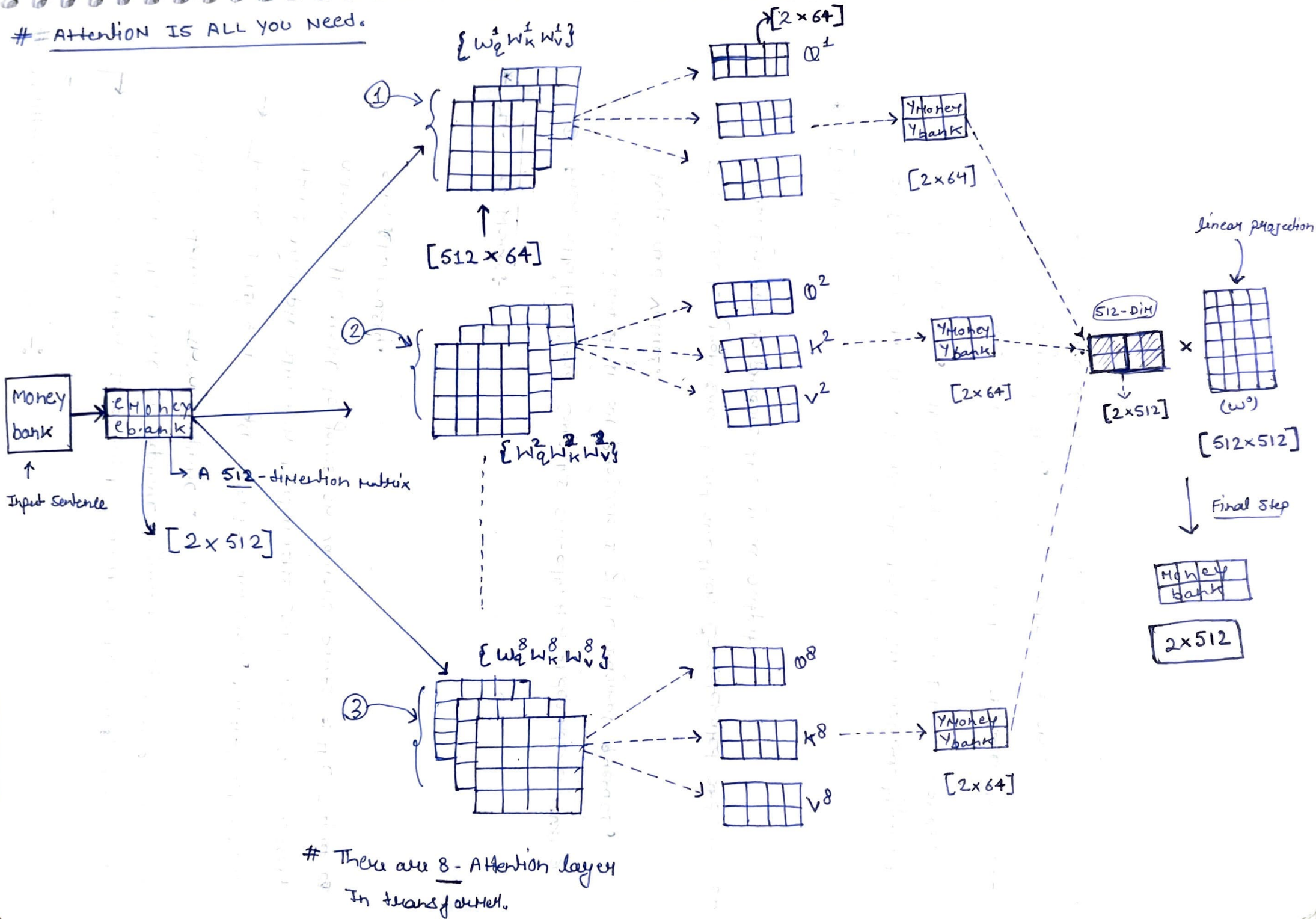
② The results from each head are then concatenated and linearly transformed to produce the final output.

* Benifits :-

① By using multiple heads, the model can capture different relationship, patterns or dependencies between words.

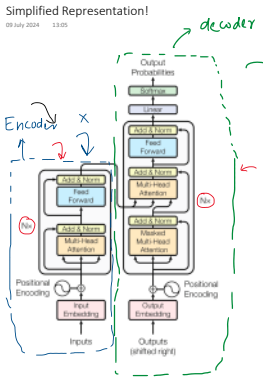
② This provides a richer representation compared to single-head self-attention, which might focus on only one aspect of the token relationships.

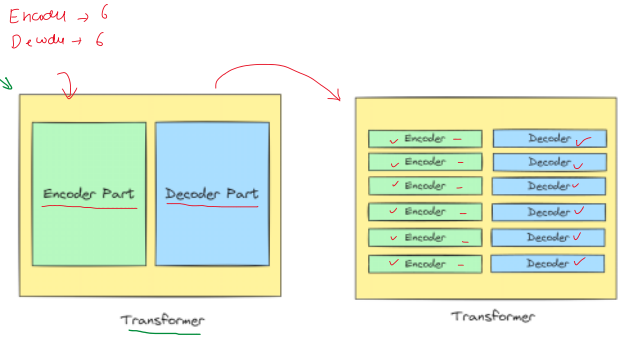
Attention IS ALL you Need.

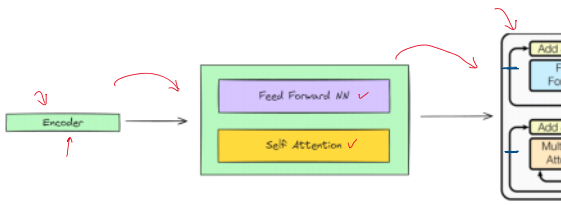


Simplified Representation!

09 July 2024 13:05

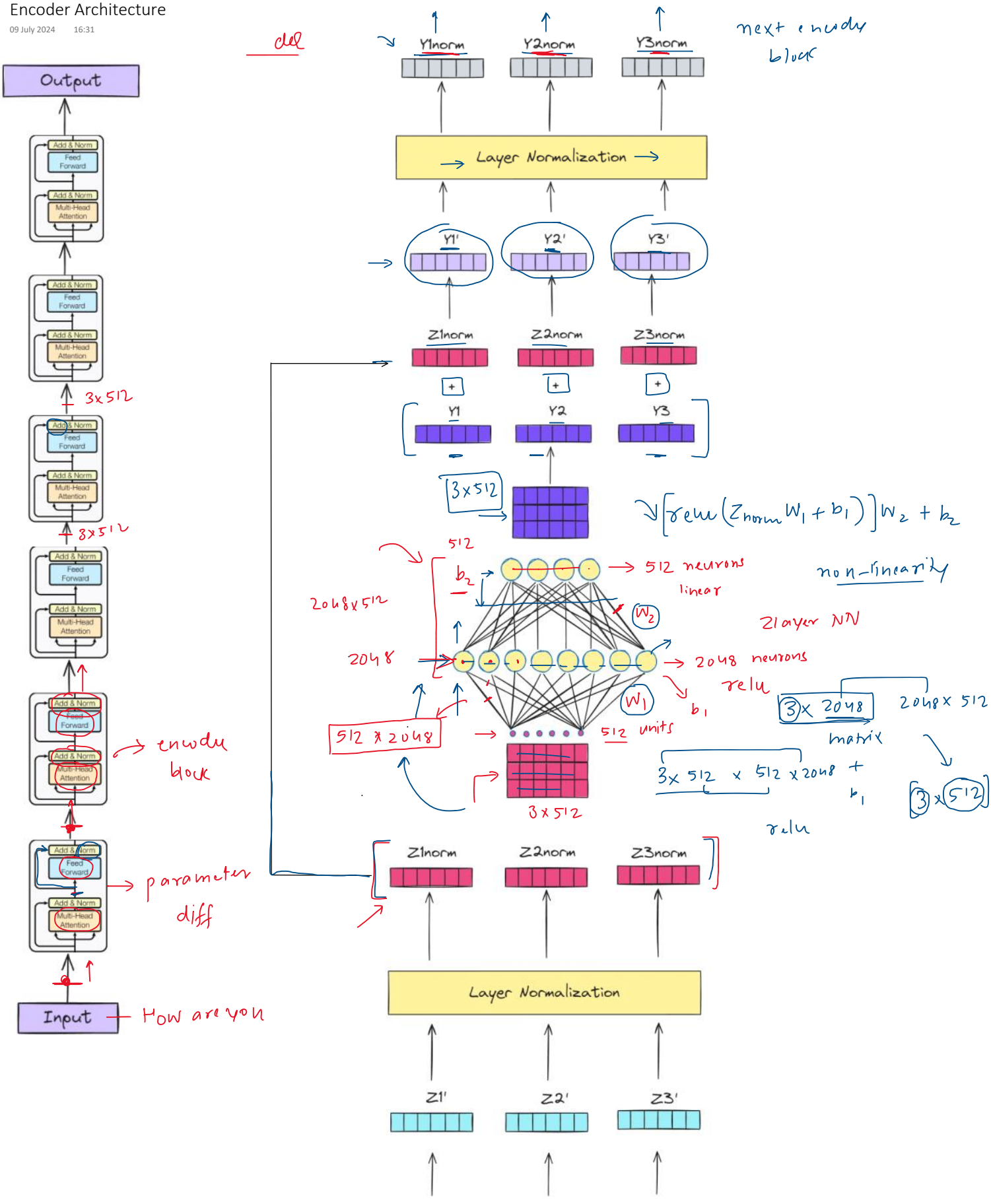


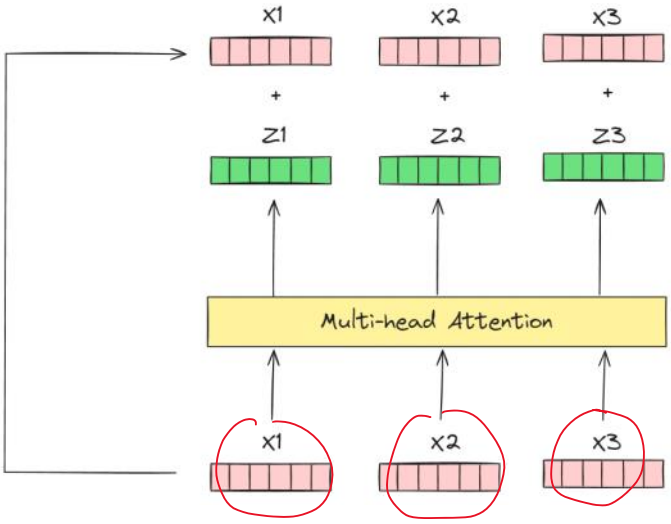


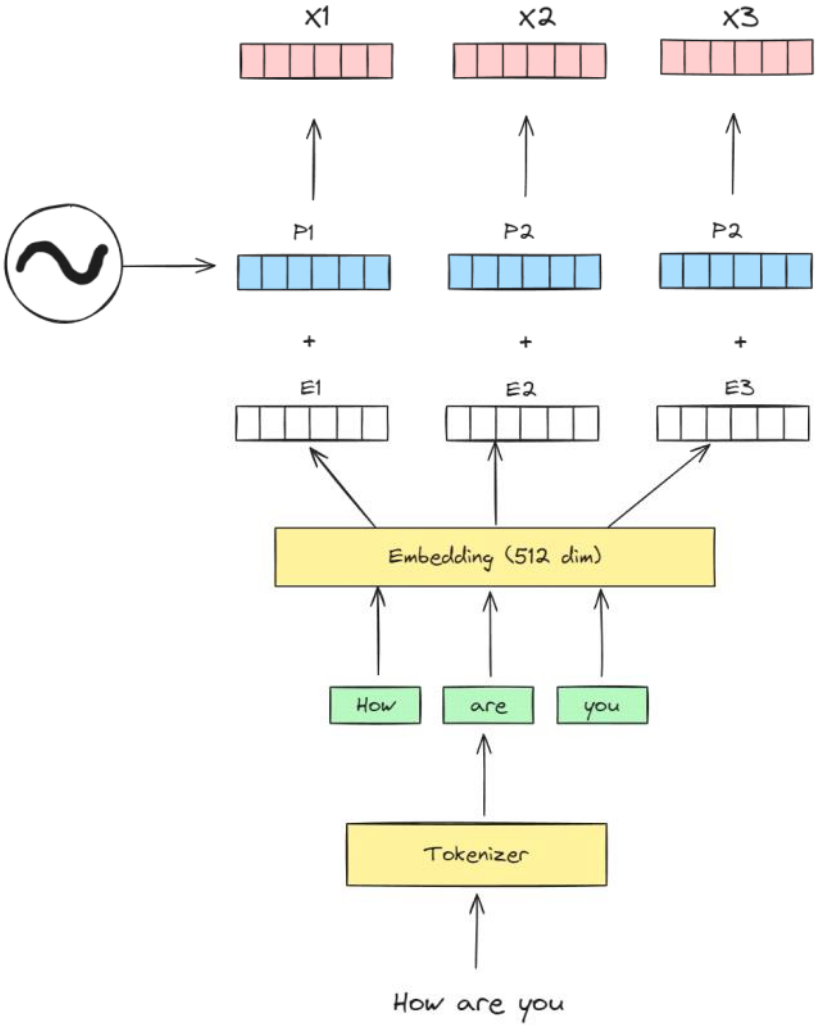


Encoder Architecture

09 July 2024 16:31




X1
X2
X3
+
+
+
Z1
Z2
Z3
Multi-head Attention
X1
X2
X3


X1
X2
X3
P1
P2
P2
+
+
+
E1
E2
E3
Embedding (512 dim)
How
are
you
Tokenizer
How are you