Transformers → chapter 2

ANN → Tabular data
CNN → Image data
RNN → Sequencial text

(Neural network Architecture)

Transformer → Sequence to sequence task

ex → Translation
→ Question - Ans
→ Text summerisation

→ parallel processing
→ can train over a big data
→ Use self - attention
→ Induction in → Attention all you need
→ In 2017

→ Impact
- Revolutionalized NLP (natural language processing)

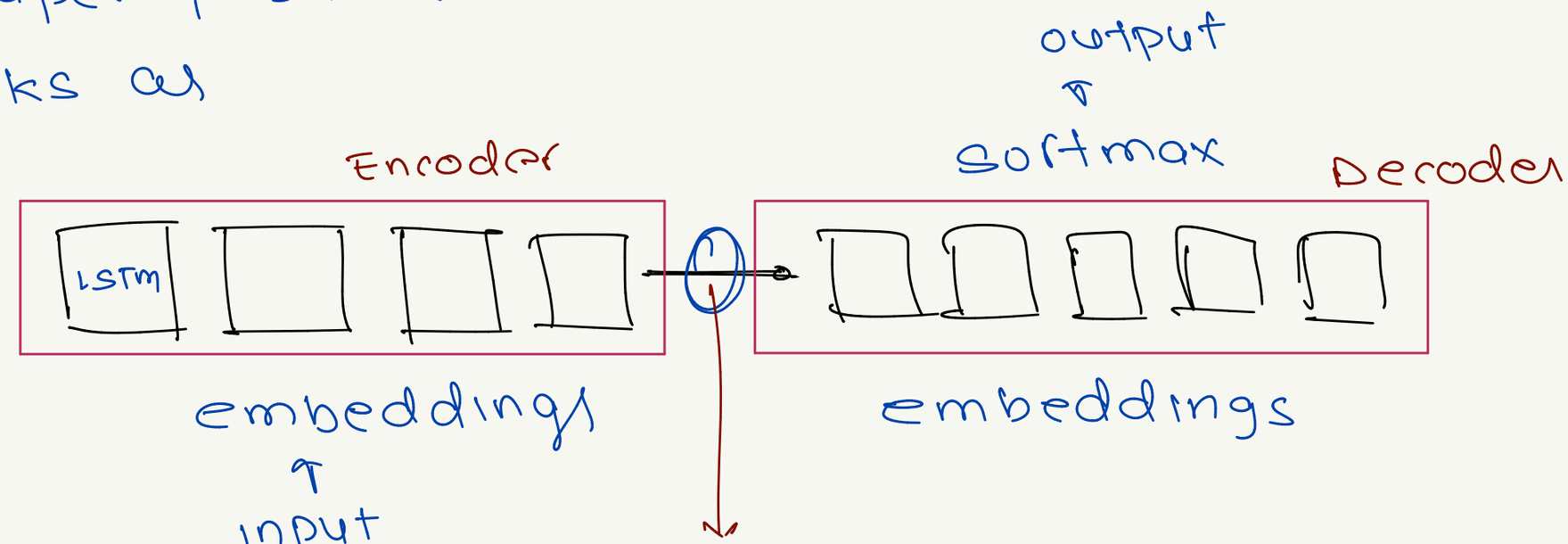
- Democratizing AI
  - ↳ BERT
  - ↳ GPT
- Multimodel compatobility
  - ↳ can take text, image etc as input.
- Acceleration of Gen AI, generated text, image etc
- Unifying of deep learning.
  - ↳ Transformer can be wed in place of NLP, Gen AI, Reinforcement.

Origin of Transformer → Papers

(1) Sequence to sequence learning with Neural network

- A paper published in 2015

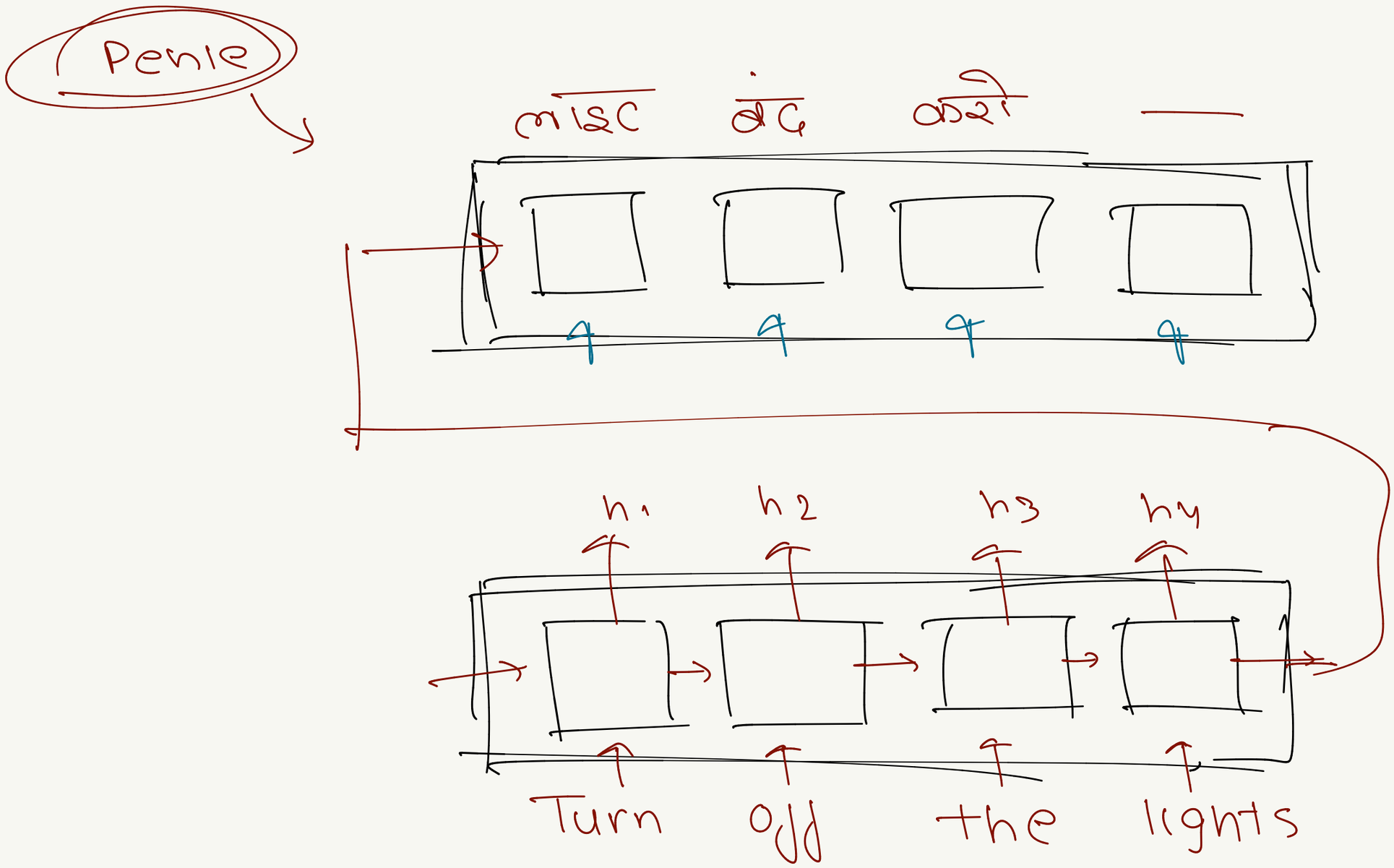
works as



- This architecture works perfectly upto 30 words but after that it fails to do properly.
- Flaw → Hum input ko context vector mein summerize karna chahte hain par agar input bht bda ho gya to context vector sab acche se store nhi kar paayega.

(2) Neural machine translation by jointly learning to Align & Translate -

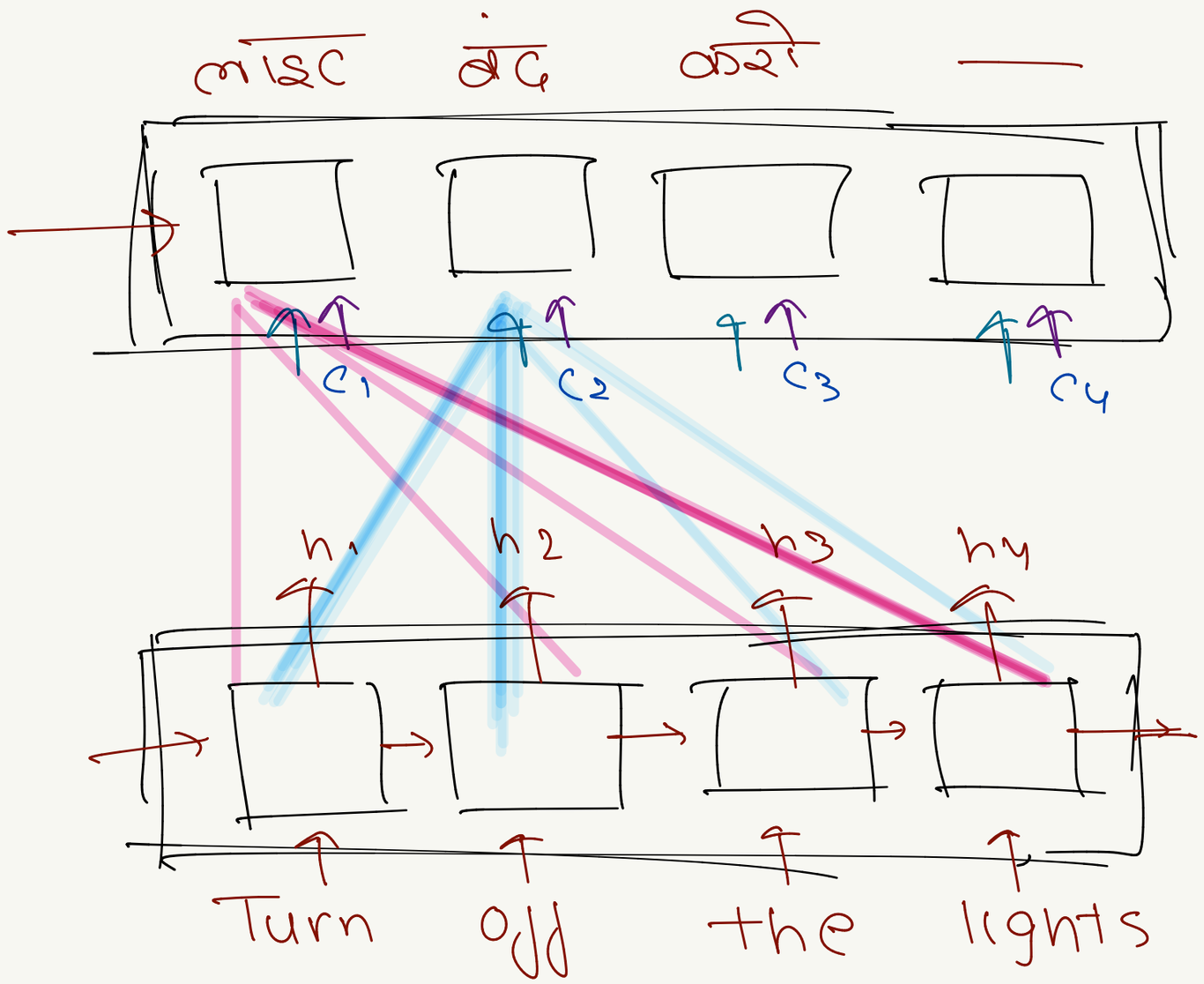
- first time introduced **Attention**



Pehle hum saara ka saara line ke context vector bhejte the decoder mein

Abb

Hume saare input ek saath bhejne ki jarroraf nhi hai.



$c_i = \sum \alpha_{ij} h_j$

Attention weights

overall context vection is weightage sum of encoder ka hidden state ka.

Attention ye batata hai kaune se input ka weightage jyada hoga kiss word ko print karne ke liye.

→ This architecture we perfectly working on even more than 30 words.

flaw → sequential training
so very slow
can't train big data set.

not parallel training
→ Bcz of ISPM

(3) Attention all you need →

→ In 2017

→ No RNN or LSTM

→ works on self-attention

→ can be trained parallely

→ stable hyperparameters

2000 - 2014 → RNN / LSTM

2014 → Attention

2017 → Transformer

2018 → BERT / GPT

2018 - 2020 → vision Transfer / Alphafold-2

2021 → Gen AI

2022 - 2024 → Chat GPT / Stable diffusion

Generally research papers are increm-ental. A small change in privious research paper

· But

This paper was totally different from all older re-search paper.

Advantages of Transformer

1) Highly scalable → parallel training

2) Transfer learning → ✓

3) multimodel Input/output → image ⇄ text
audio ⇄ text
image
etc

4) Flexible architecture
↳ can change small things can create our own architecture
BERT → encoder only

5) Ecosystem → many library (tagging face)
& tools
& Blogs & Vedios

6) Eaily can be integrated in other AI.

Applications

(1) chat GPT (2) DALL-E2 (3) Alphafold-2

④ OpenAI codex ⑤ Github copilot etc

Disadvantages

① High computational tools
② Data (reawres lot of data)
③ overjitting (maybe)
④ Energy consumption
⑤ Interpretability (its a blackbox model)
(not secure)
⑥ Bias → Ethical concerns

Self Attention

Q) What is self attention?

for NLP ⇒ words → Numbers (vectorization)

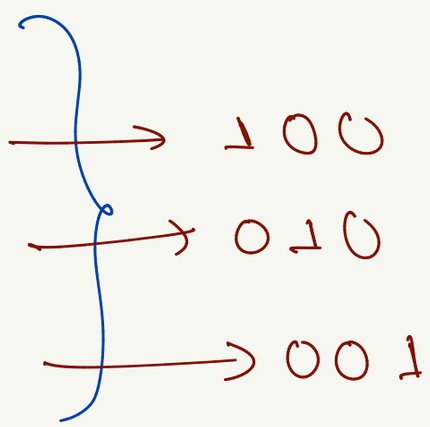
computer only understands numbers

- OHE → One hot encoding
  - → understands unique words

Example → 1) cat, Rat, cat

2) mat, mat, Rat

	mat	rat	cat
mat	1	0	0
Rat	0	1	0
cat	0	0	1



So numerical representation of

(cat Rat mat) → [100][010][001]

- **BOW** (Bag of words) → Also counts number of occurance.

Example

1) cat mat mat

2) cat rat rat

BOW of 1) → [cat Rat mat]
↓ ↓ ↓
[1 0 2]

BOW of 2) → [cat rat mat]
↓ ↓ ↓
[1 2 0]

⇒ **word embeddings**

↳ have power to capture semantic meanings
what is the contex & how generally this word is wed

How to create?

→ collection of large amount of training data
→ Sent to neural network
→ It understands meaning & context
→ & represent each word into n-dimentional vector. (n can be 64, 256, 512 etc)
→ After this we can reprsent words into set of 5 numbers.

Example

King = [0.6 0.5 0.2 0 0.9]
Queen = [0.7 0.6 0.9 0.01 0.5] } Similar

Mouse = [0.5 0.9 0.6 0.08 .9] } Different

So, jitna similar words honge unke vectors utne similar honge.

- vector ke har dimention ek parameter ko show karte hain.

Example

Queen = [0.9 0.5 0 0.2 0.8]

0.9 → Rolality
0.5 → Humon
0 → male
0.2 → poor
0.8 → Beautiful

Just every dimention has a meaning.

- Embedding → word meaning → converts into vector

Problem

Example →

1) An apple a day keeps doc away.
2) Apple is healthy.
3) Apple is better than orange.
4) Apple moks good phones.

we will be representating all the words into 2D [x, y]

[x, y] → [Taste, Tech]

word embeeding capture → Average meaning

for sentence

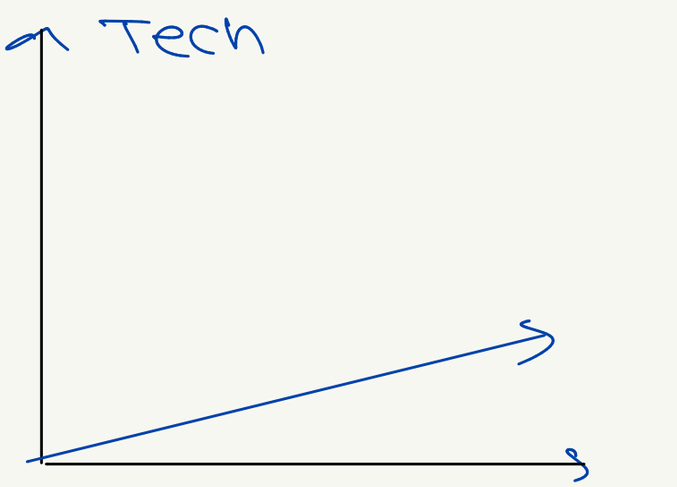
1. [0.6 0]
2. [0.7 0]
3. [0.8 0]
4. [0.8 0.2]

lets assume out of 1000

900 → Taste

100 → Tech

overall vec → [0.9 0.3]



overall word embeding depends on data set.

→ word embeding sirf ek baar bante hain phir static ho jaate hain for every further we.

Example

Apple launched a new phone while I was eating.

convert this int hindi.

→ [0.9 0.3] → It will take as food
→ while it should be used as phone

so, instead of static embedding, it should be → contextual embedding.

This problem is solved by **self-attention**

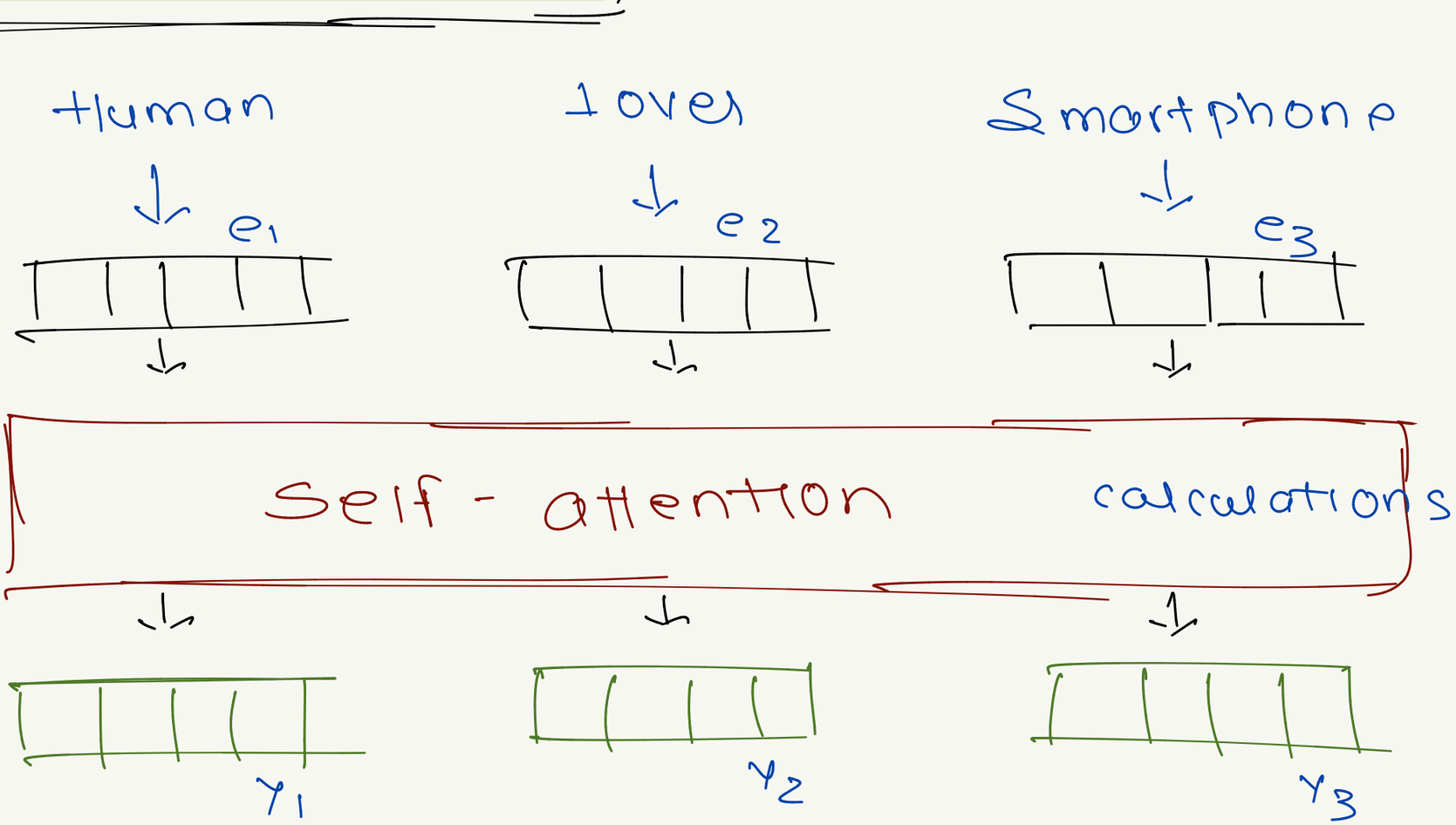
It creates new embeddings based on context.

does not use static embeddings.

→ This is used in transformer.

how do self-attention works –

Example



money bank grows

bank → 0.3 money
+ 0.7 bank
+ 0.1 grows

money → 0.7 money
+ 0.2 bank
+ 0.1 grows

grows → 0.1 money
+ 0.2 bank
+ 0.7 grows

River bank flows

bank → 0.5 river
+ 0.4 bank
+ 0.1 flows

river → 0.8 river
+ 0.15 bank
+ 0.005 flows

flows → 0.4 river
+ 0.01 bank
+ 0.59 flows

we need to change money, bank, grow into embeddings too.

$$\text{new embedding of money} = 0.7(e_{\text{money}}) + 0.2(e_{\text{bank}}) + 0.1(e_{\text{grows}})$$

(nD) (nD) (nD) (nD)

Iska matab ye hai ki new embedding of money

0.7 time (money ke poorane embedding)

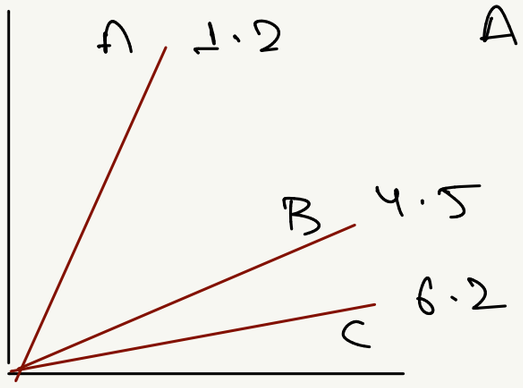
+0.2 times (bank " " ")

+0.1 times (grow " " ")

se bna hai. (Representing similarities)

- Dot products shows similarity

for example



A.C = 6 + 4 = 10

B.C = 24 + 10

= 34

As B & C are more similar to each other than A & C.

So the no.s in dorant of embeddings are actually

DOT PRODUCTS

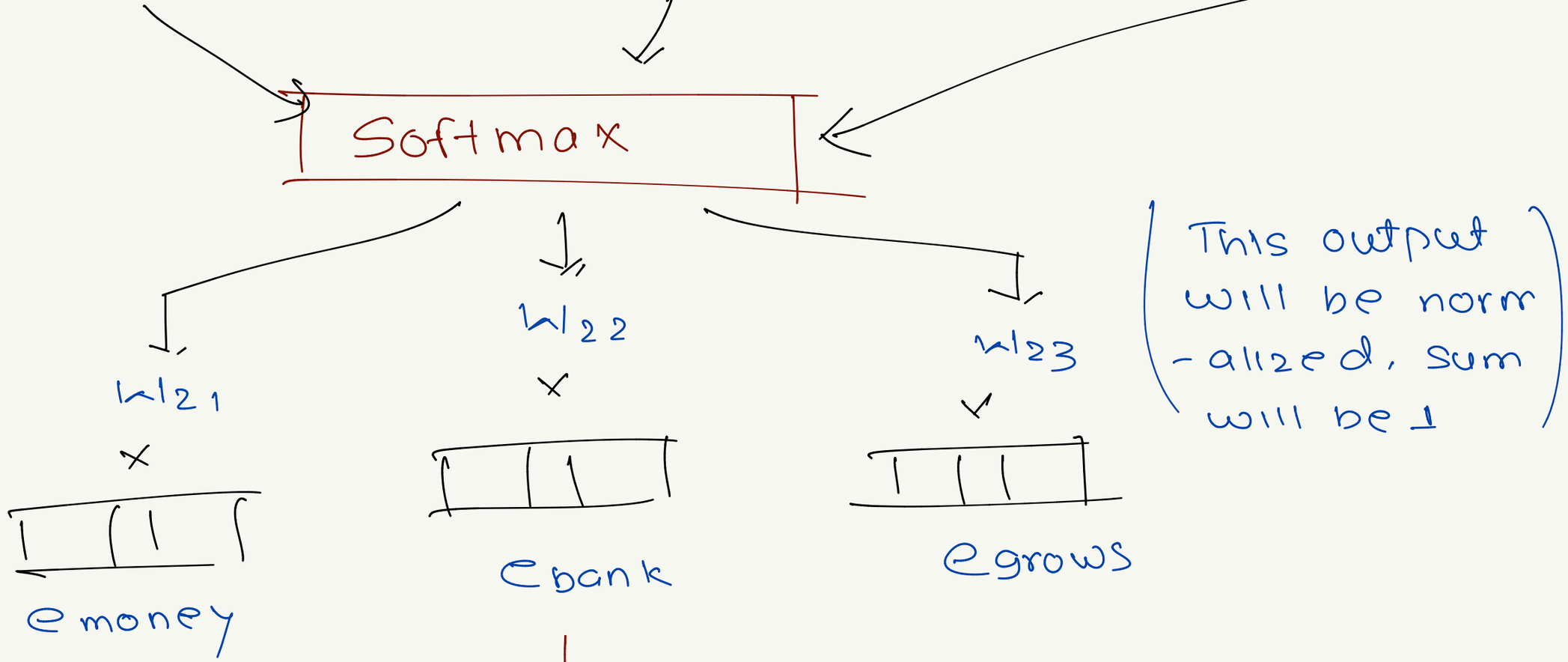
So, we can write this eqn →

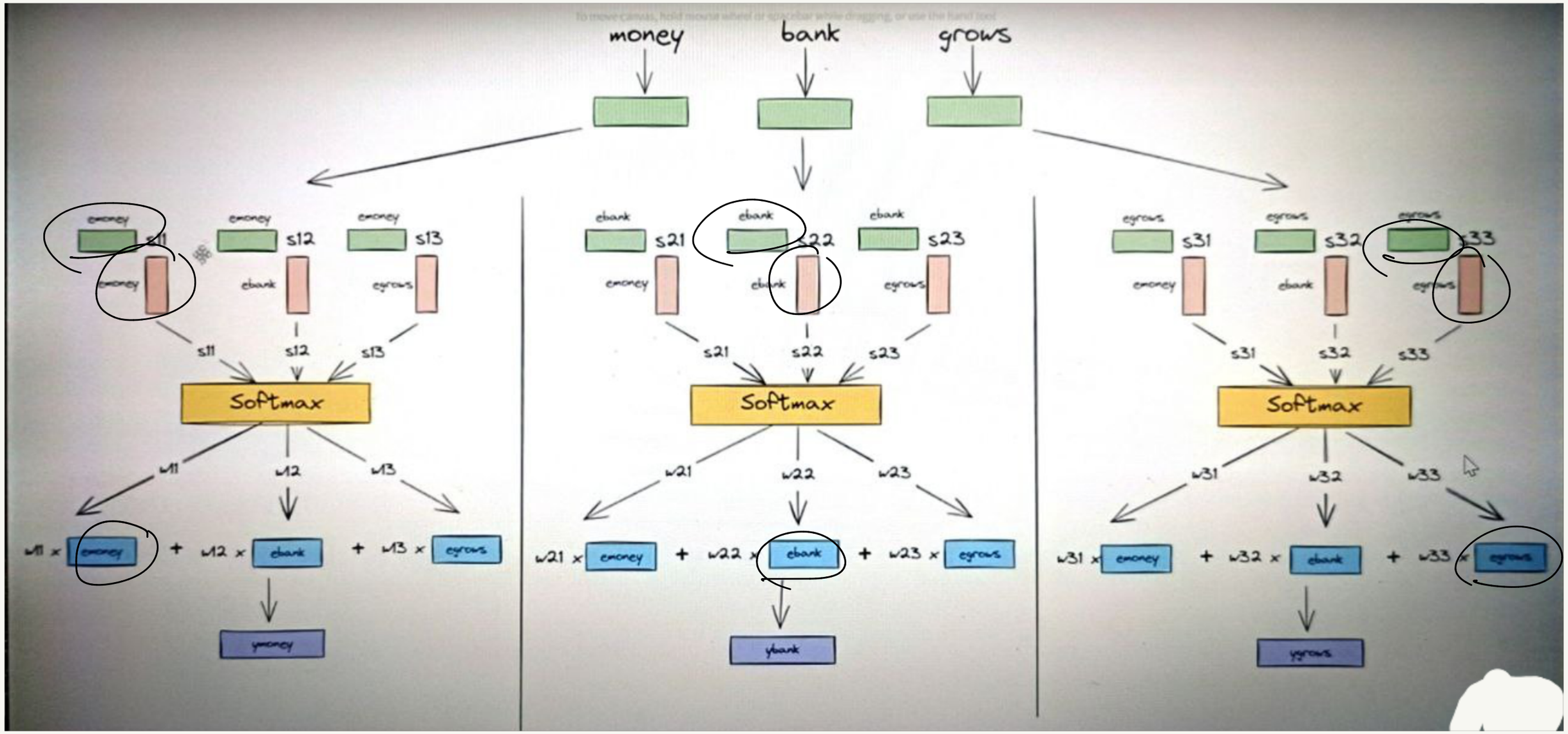
bank → 0.3 money
+ 0.7 bank
+ 0.1 grows

into → New embedding of bank = [e_bank • e_money] e_money
+ [e_bank • e_bank] e_bank
+ [e_bank • e_grows] e_grows

visual of dot product →

Dot product


Softmax
w_{21}
w_{22}
w_{23}
e money
e bank
e grows
This output will be normalized, sum will be 1
e bank new
→ contextual word embeddings
→ weightage sum



- This can be done paralley.
  (even there are 1000 words)
  we can generate contexual vector.

Disadvantage → looses sequential order.

→ There are no learning parameters.

means its not task specific, it just develops general contexual embedding.

Example

Piece of cake ⟶ but we want it as

केक का टुकड़ा

(Generalized)

→ failed

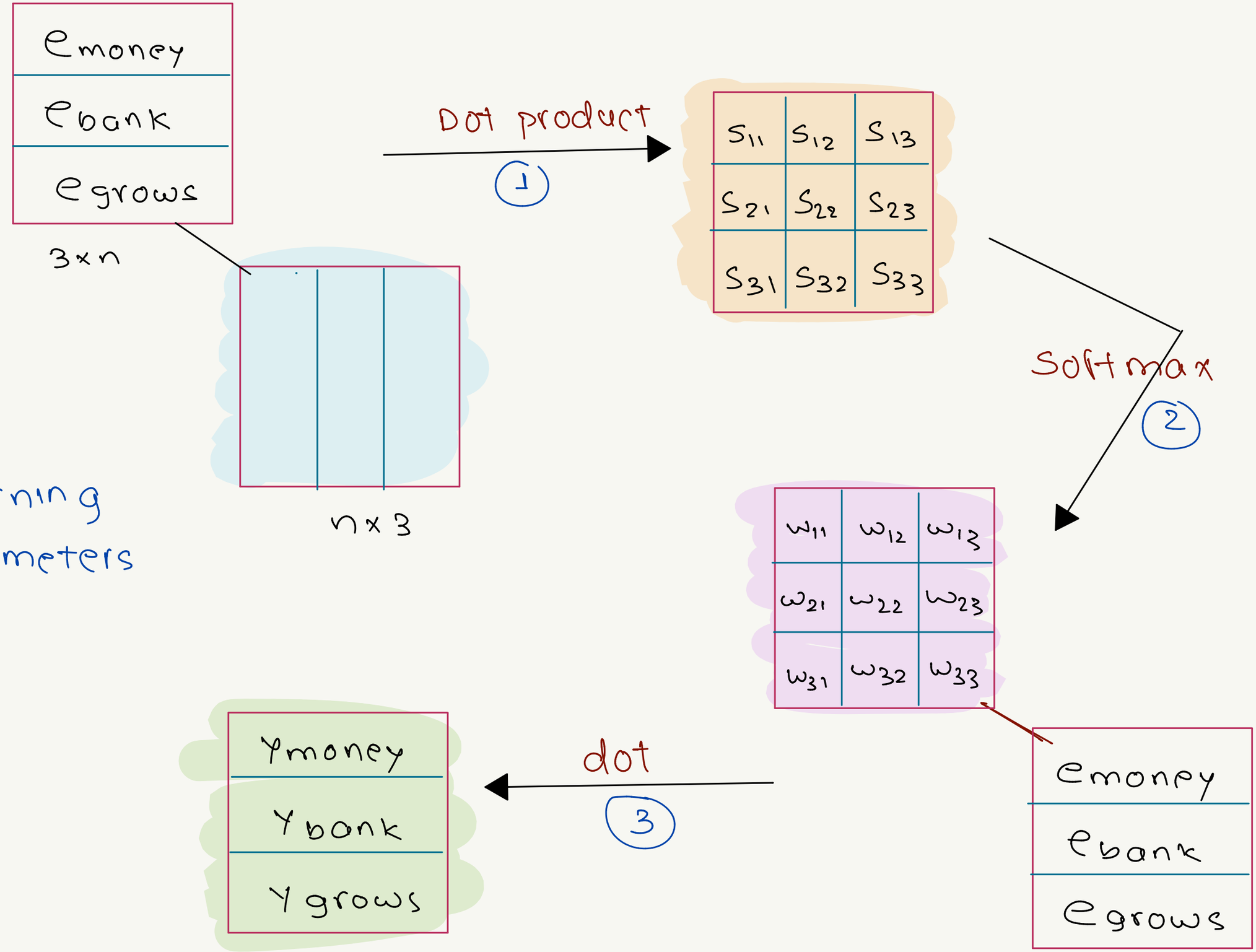
Break a leg ⟶ शुभकामना

तोड़ दो

Overall → only contexual embedding is not perject we need task specific contexual embedding

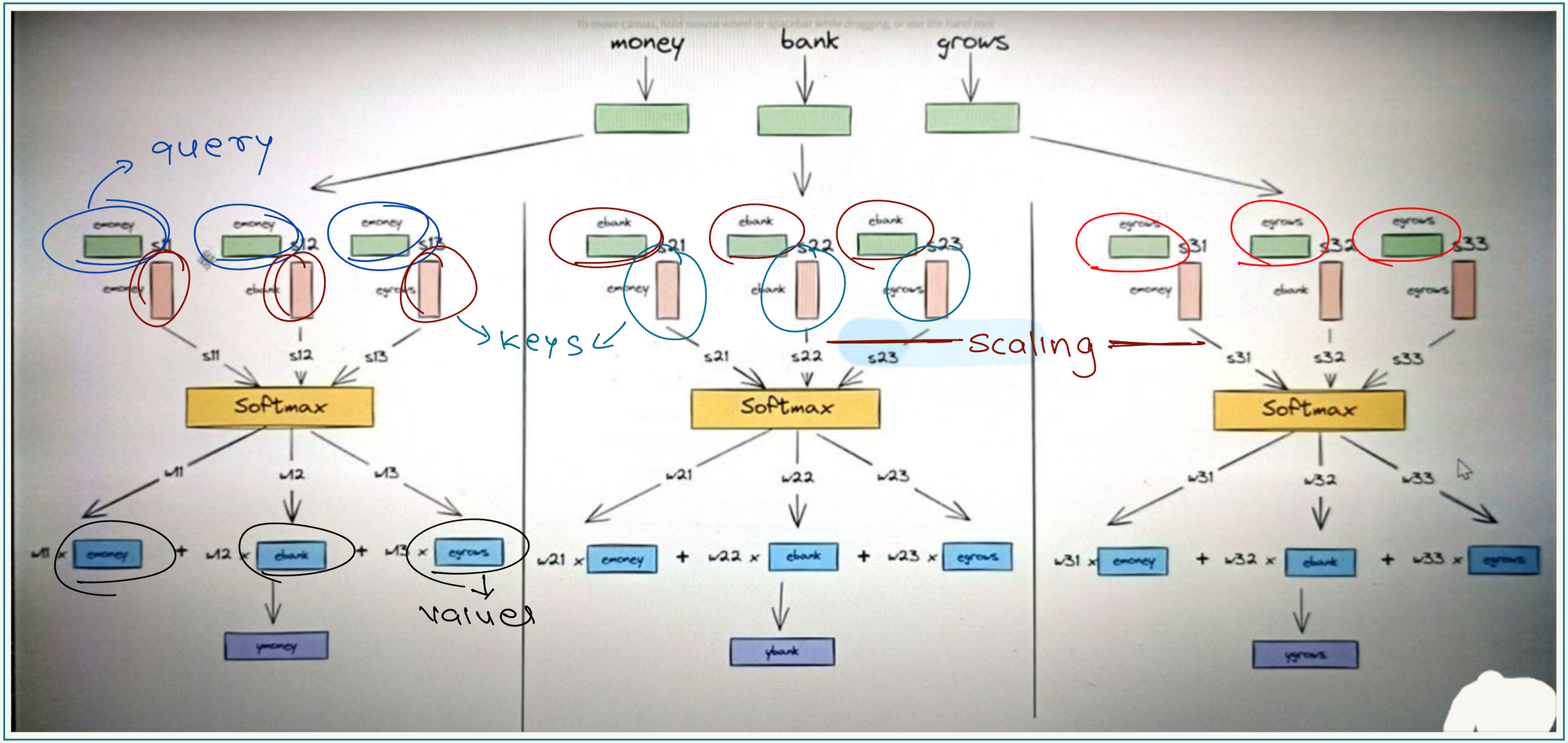
General contexual embedding → Add weights & Bias → Task specific contexual embedding.

parallel embedding -



No learning parameters

we can add weights in step① & ③ only.



Lets understand with the help of dictionary-

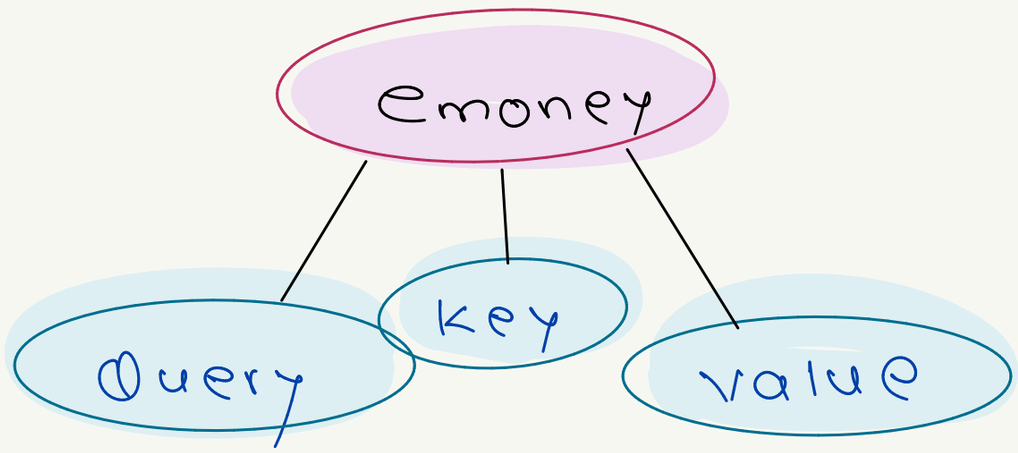
$$d = \{ a : 2 \;, b : 3 \;, c : 4 \}$$

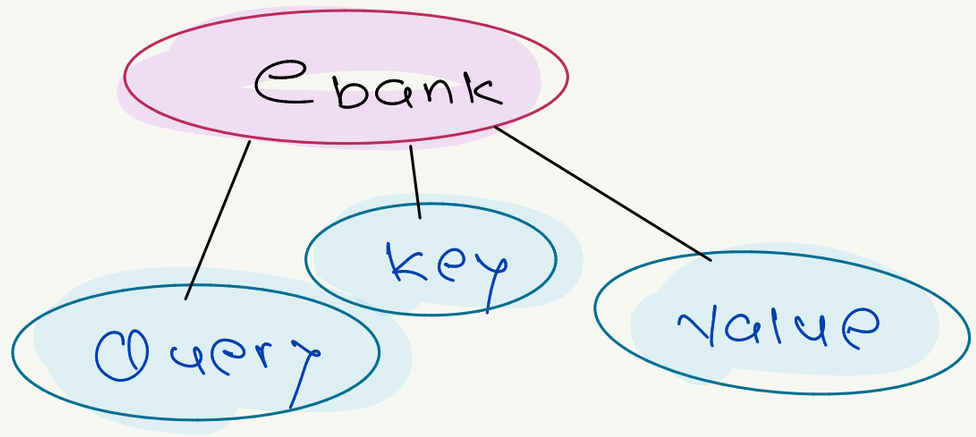
a : query, 2 : value

query key

if there is key named a, it will return value 2

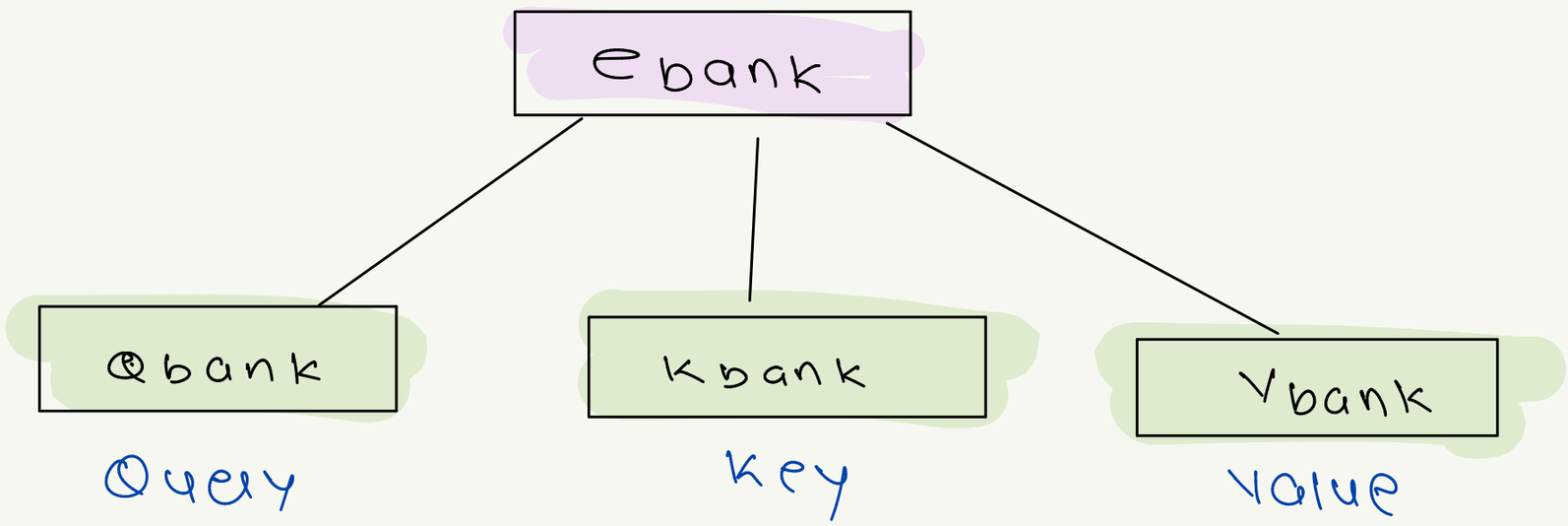
- Each of our word embedding is used as 3 ways



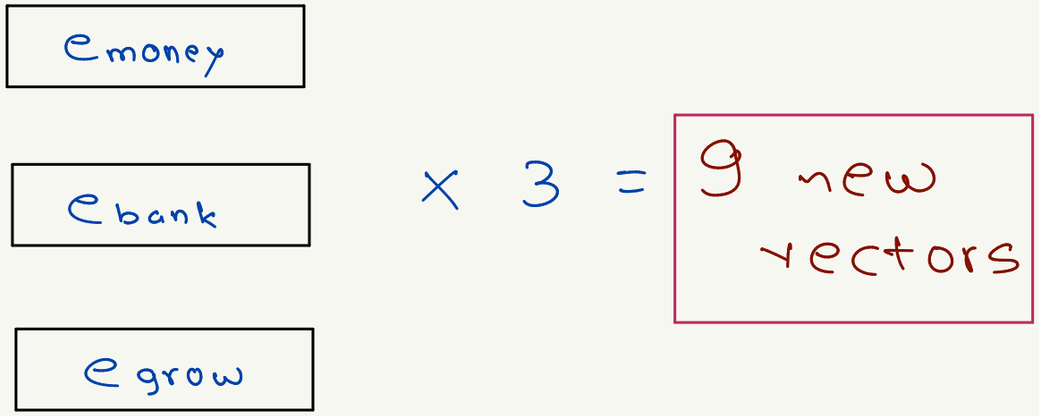


- So, we need to specify thing, we can't use full word embedding as query, key & value.

- This Query, key and value vector will be formed with the help of data & we.

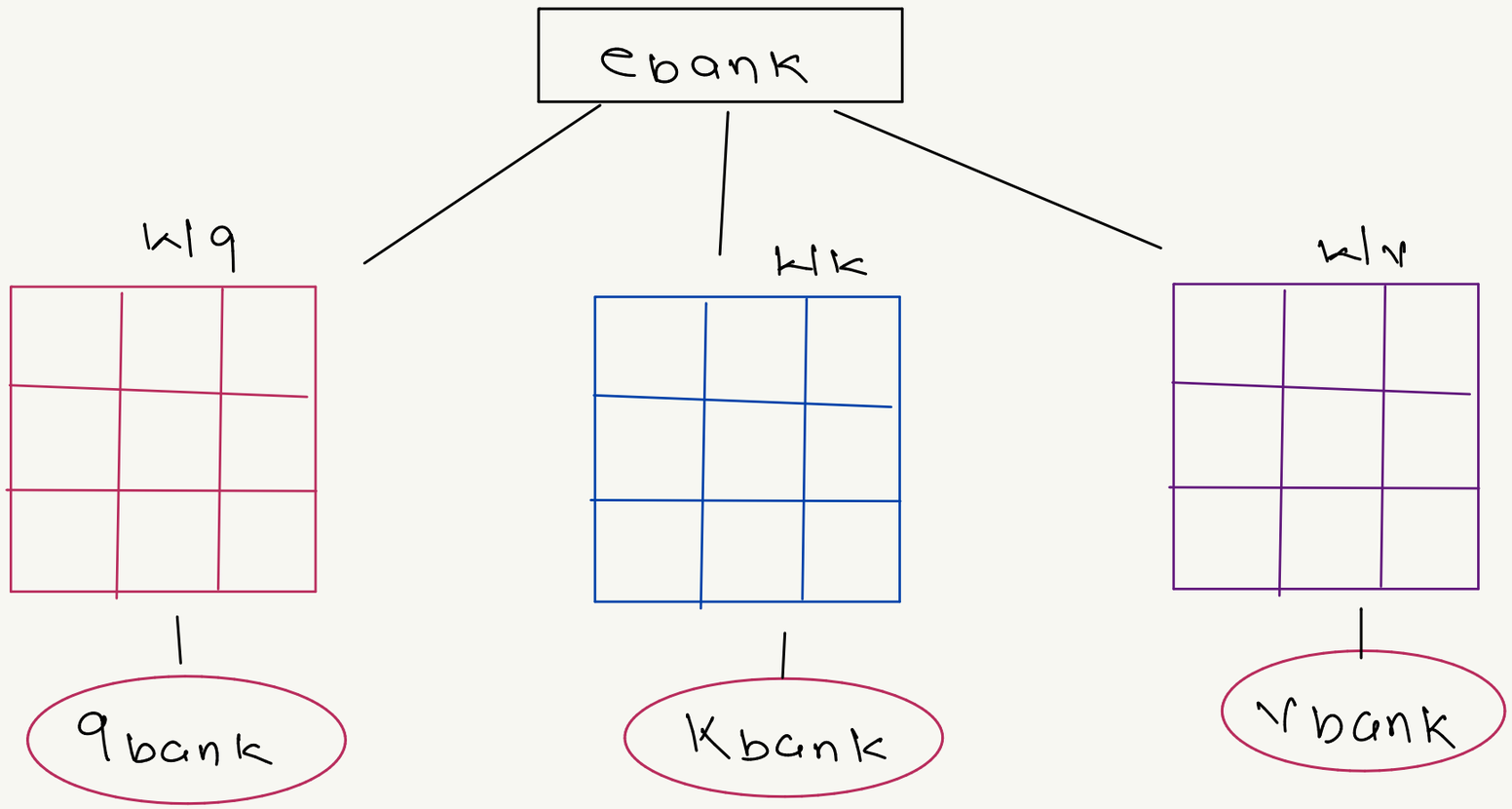


So, we have 3 words



→ By linear transformation

→ By multiplying original vector with diff matrix to get 3 new vectors.

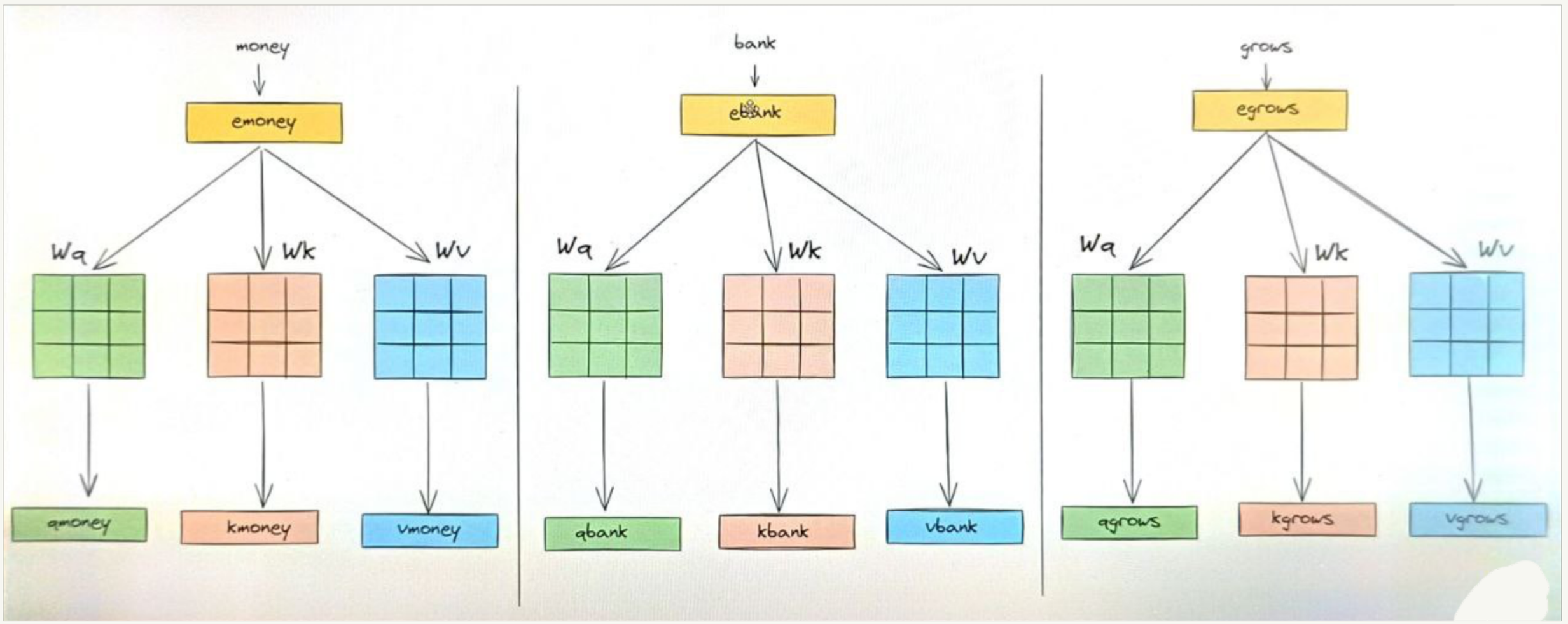


matrix will be decided by training process & data.

↓

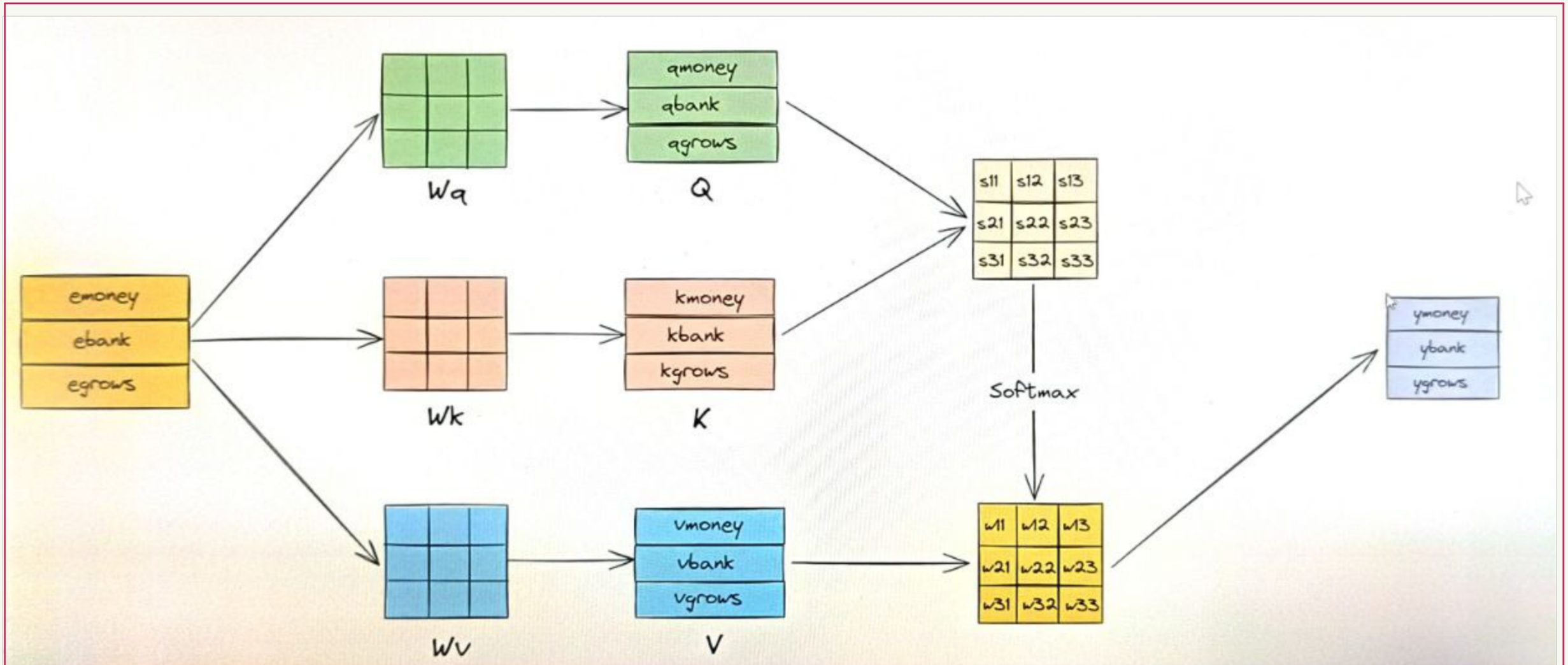
we will Start with random weights

Ye matrix baar baar training karke sni value gain kar lenge,



$w_{q1} = w_{q2} = w_{q3}$ & $w_{k1} = w_{k2} = w_{k3}$ & $w_{v1} = w_{v2} = w_{v3}$

& we finally got task contexual embedding



we can perform all the action parallely with the help of powerful GPUs.

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

Scaled dot product Attention

d_k → dimentionality of k vector.

→ dimention of key vector.

to reduce unstable grading.

So, sortmax lagane se pehle hum saare value ko $\sqrt{d_k}$ se divide karna padega.

kiny? → Dot-product ka nature

$\underbrace{Q \cdot K^T}$

Dot product of 2 matrix

→ Actually we are doing dot products of multiple vectors.

when we have low dimentional dot product

↳ low variance

when high dimentional dot product

↳ high variance

problem

Softmax → converts given data into probability & normalize it.

→ depends on e^x → so bde no ko very high probability dega

↳ aur chhote wale ko bht low.

so, overall scaling se chote aur bde no dono proper traning hogi, kyoki scaling se variance kam hoga.

Example

1) ([10, 20, 30, 40, 50, 60, 70])

var = 400

2) $\left(\left[\frac{10}{10}, \frac{20}{10}, \frac{30}{10}, \frac{40}{10}, \frac{50}{10}, \frac{60}{10}, \frac{70}{10}\right]\right)$

→ ([1, 2, 3, 4, 5, 6, 7])

variance → 4.0

(Spread bw no. in dataset)

Multi-head Attention

Example

The man saw the astronomer with a telescope.

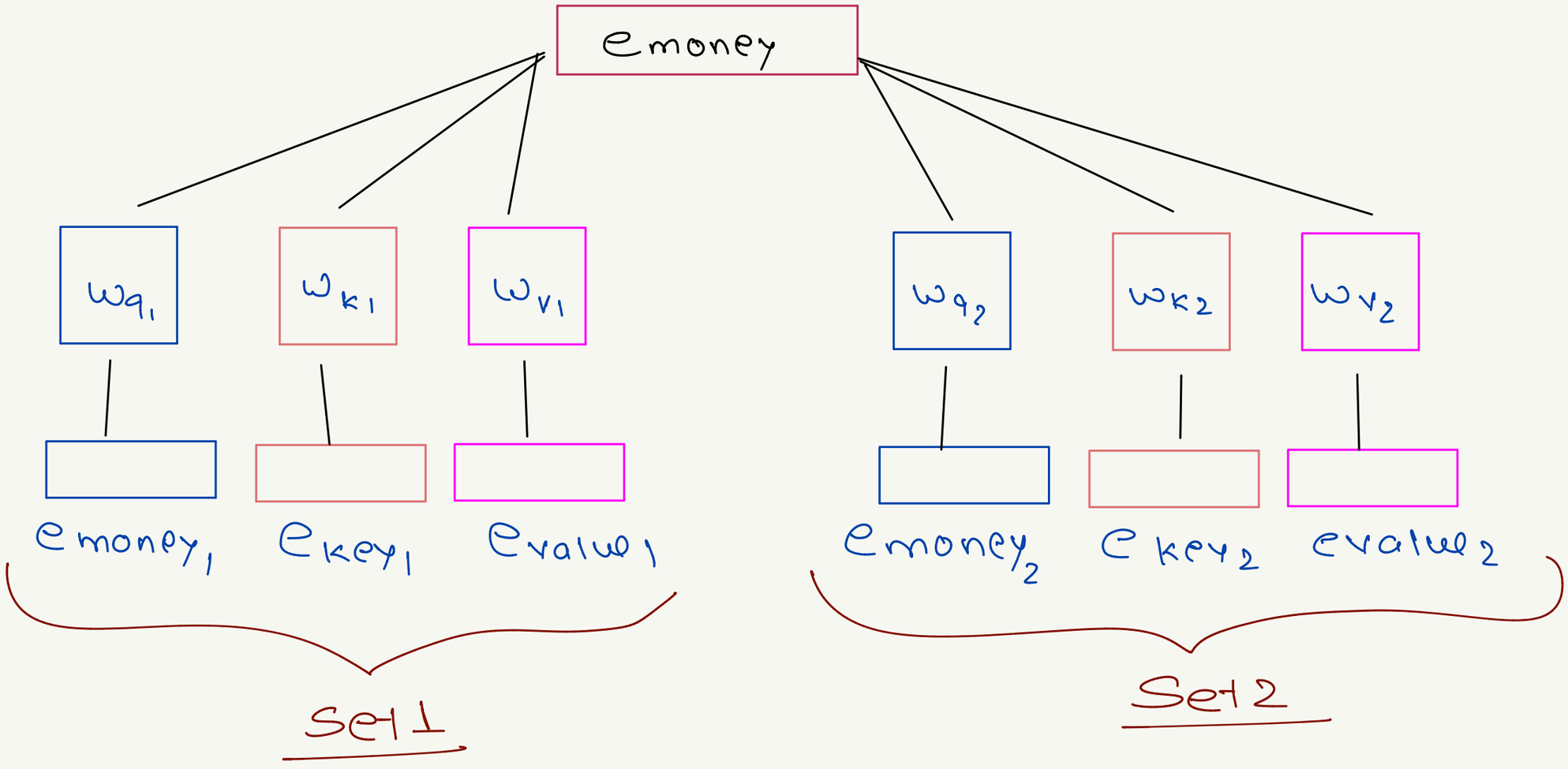
→ have two meaning, but self-attention can only generate one meaning

→ self attention can only consider single perception.

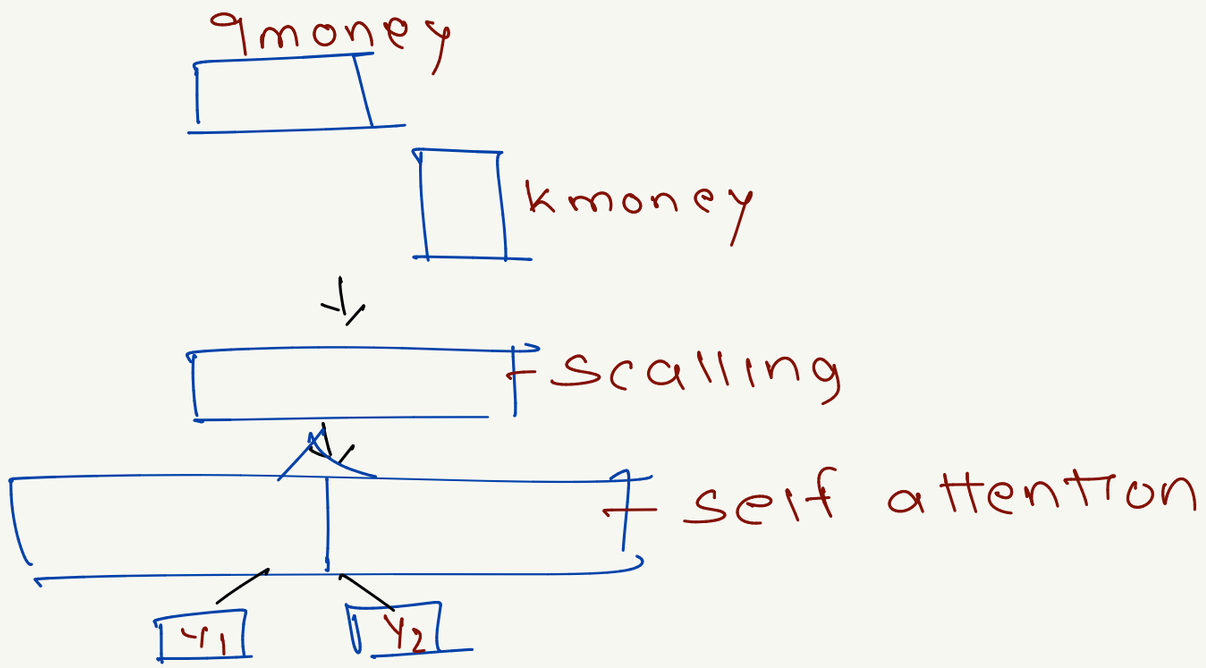
⇒ Multihead attention solves this problem.

↳ ek se jyada self attention use hoga.

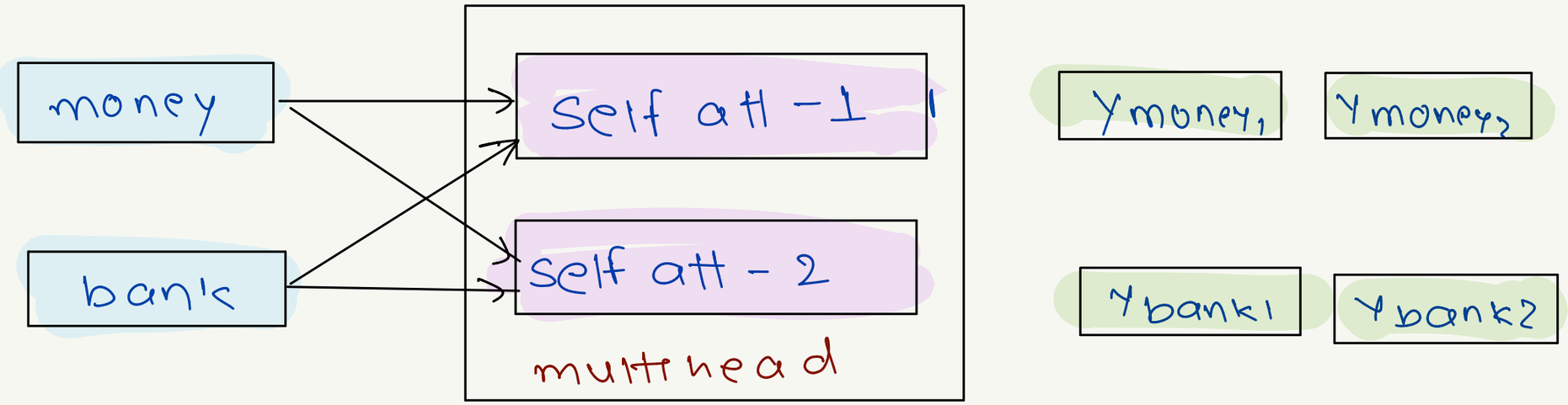
Example



- Aise hi bank & grows ke liye bhi 2-2 set aayenge



2 contexual text vector aayega,


money
bank
self att - 1
self att - 2
multihead
Ymoney1
Ymoney2
Ybank1
Ybank2

Positional Encoding

- we can generate contexual embeddings parallely by self-Attention.
- order of words lost

For self attention

1) Harshit killed lion.
2) lion killed Harshit.

Both are same for self-attention.

Sol (1) If we add one more dimention that will carry positional no. of word.

Prblm → If we want to procm a book than last no will be a very big no. 10000 or more

∴ Neural network hates large numbers

it prefers no bw -1 to 1

1 no. 1,2,3... are discrete nos while neural network prefers smooth transitions / Contineous.

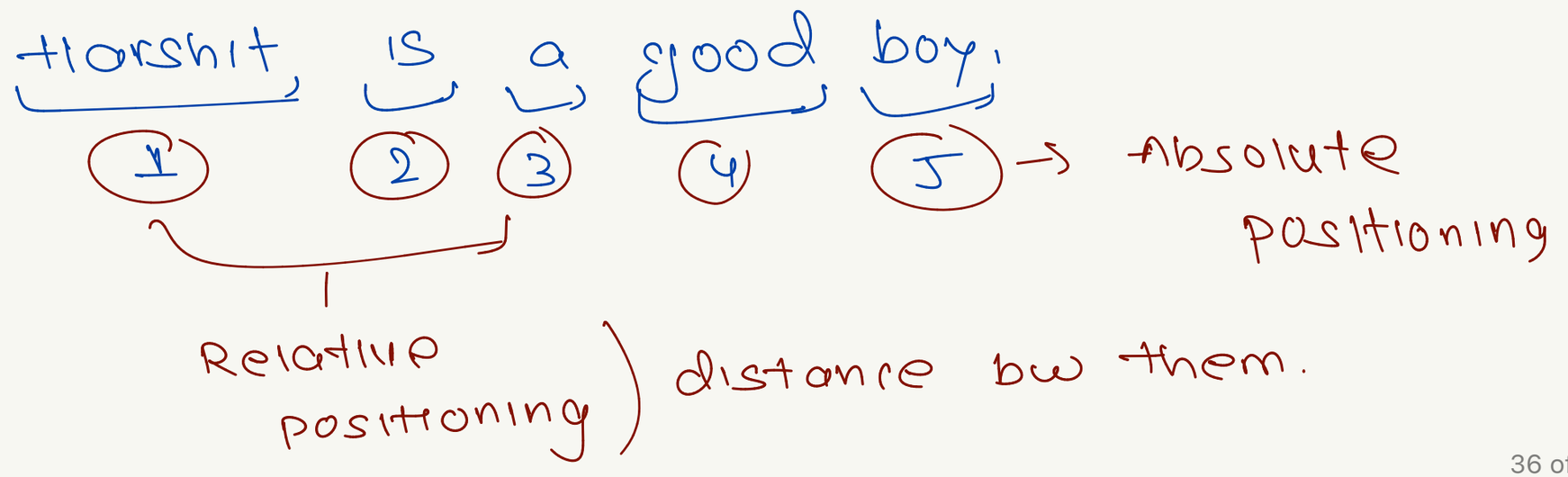
we also have sol. for this.
we can divide all the positional no. with the last no, so The no will be within -1 to 1.

Pblm

we can't normalize bcz neural network will conjure → wether the second position is 2 or 0.2 or 0.00002. Position no. will not be consistant.

Pblm 3 → we can't capture relative positioning

Ex



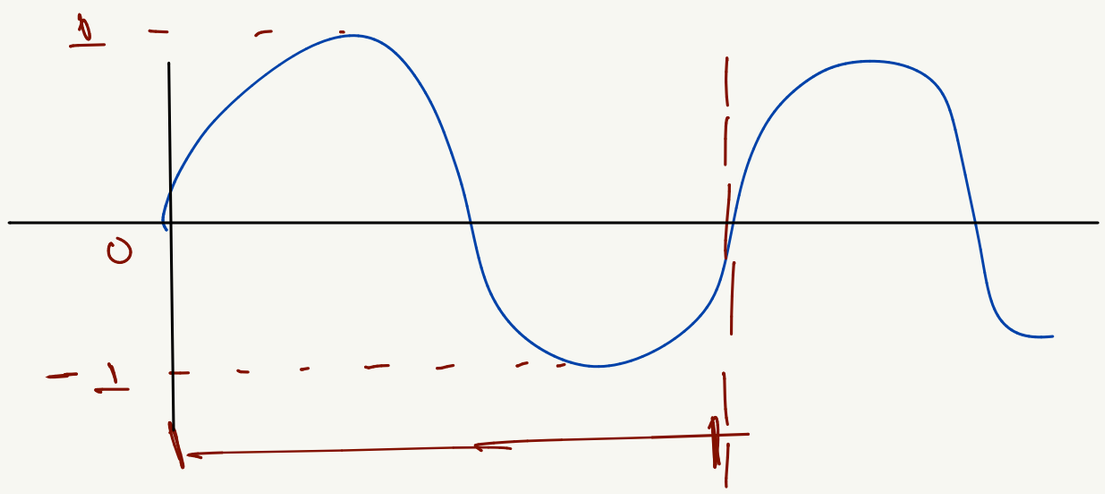
overall

1. Unbounded
2. Discrete
3. No relative position

Bounded
contineous
periodic fun

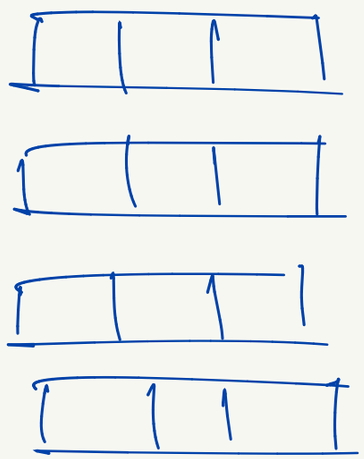
All these creteria is fullfilled by trigonometric fun.

- Sin fun
  - contineous
  - priodic
  - Bounded



Example

1 Harshit
2 killed
3 a
4 Lion



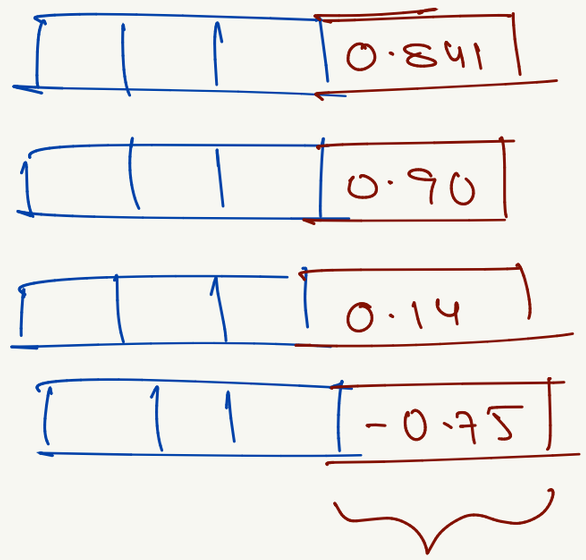
Sel
f
Athe
ntic
n

$y = \sin(\text{position})$

Summary:

- Sin(1) = 0.841
- Sin(2) = 0.909
- Sin(3) = 0.141
- Sin(4) = -0.757

1 Harshit
2 killed
3 a
4 lion

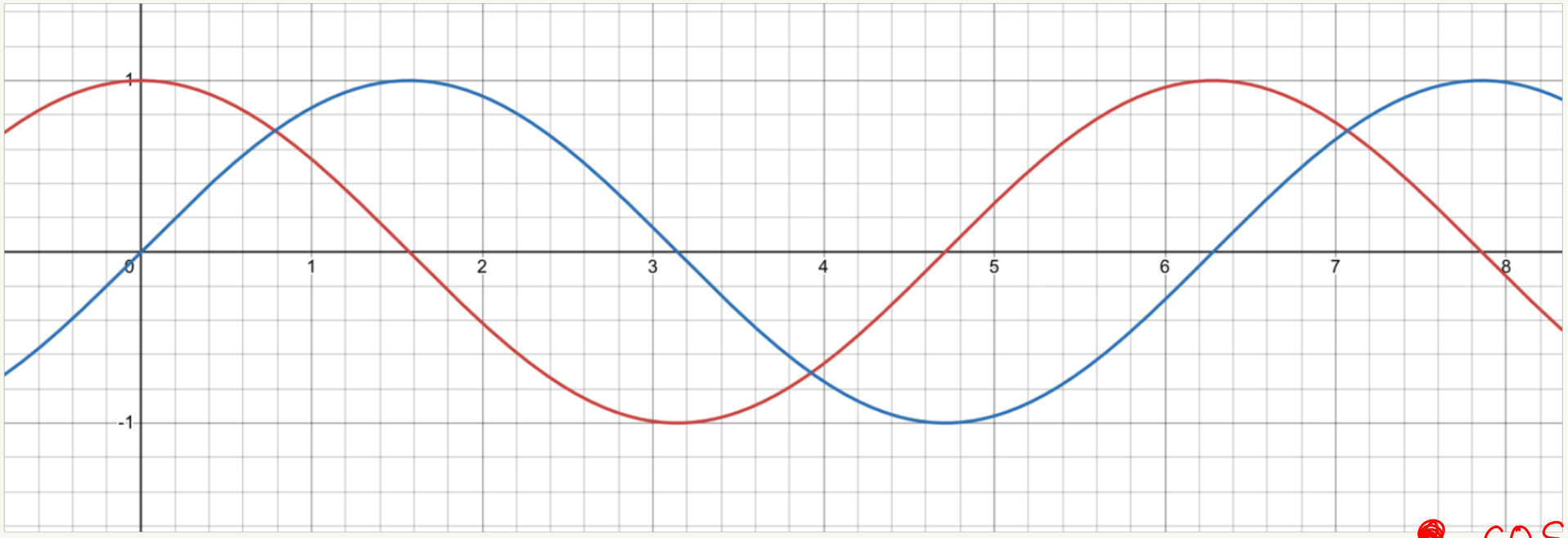


Pblm

2 word ke positional encoding same nhi ho sakta, par sin curve is periodical so ye repeat karega & 2 words ke same encoding dega.

Sol Hum ek jgh we karne ke badle 2 trigonometric jgh we kar sakte hai both sine & cos.

& position encoding → vector



cos

sin

So, now we have 2 values for each no.

Sine Values

- Sin(1) = 0.841
- Sin(2) = 0.909
- Sin(3) = 0.141
- Sin(41) = 0.656

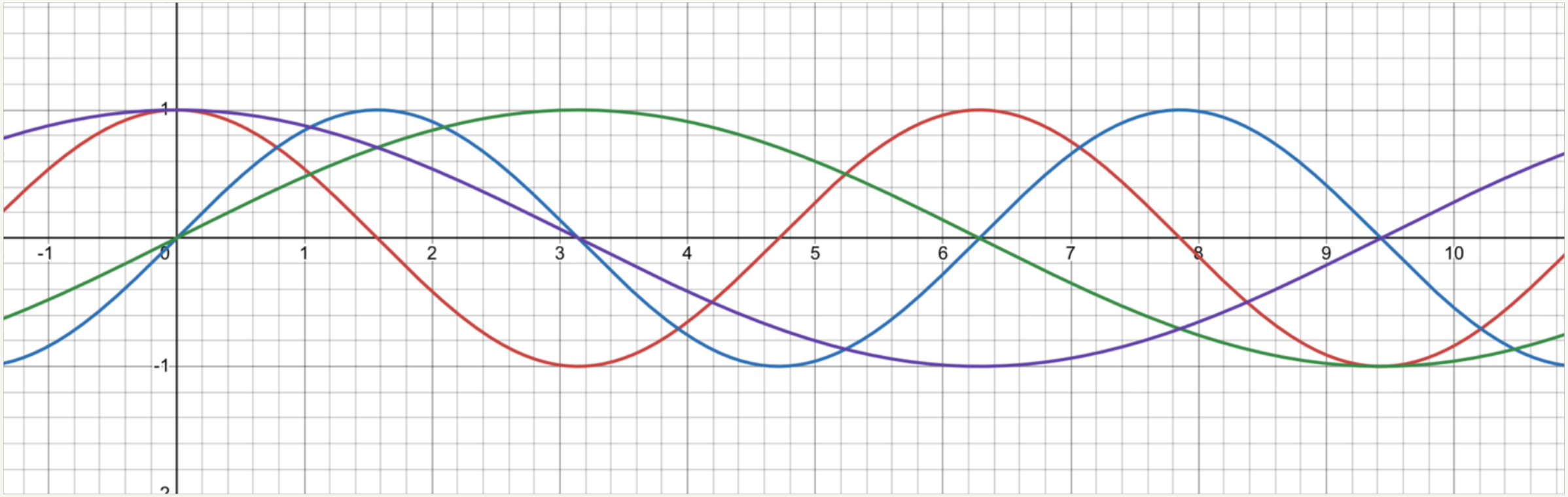
Cosine Values

- Cos(1) = 0.5403
- Cos(2) = -0.4161
- Cos(3) = -0.9899
- Cos(4) = -0.6536

Harshit killed a lion

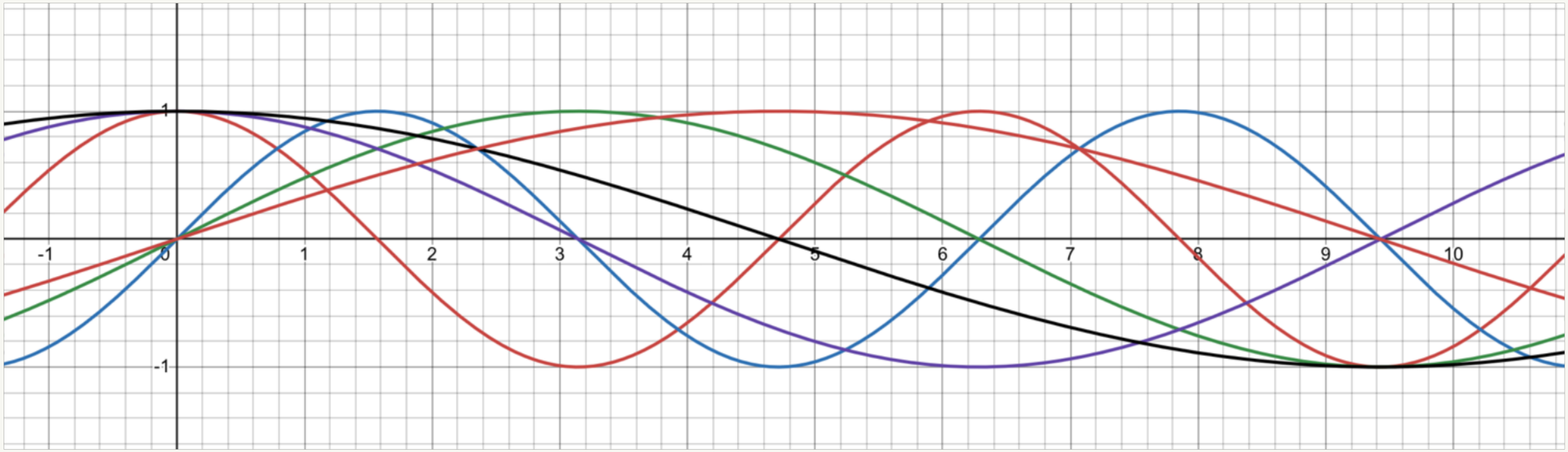
0.8	0.5
0.9	-0.4
0.1	0.9
0.6	0.6

There are still posibilites of maching, so we can add more puns cos(position/2) & sin(position/2)

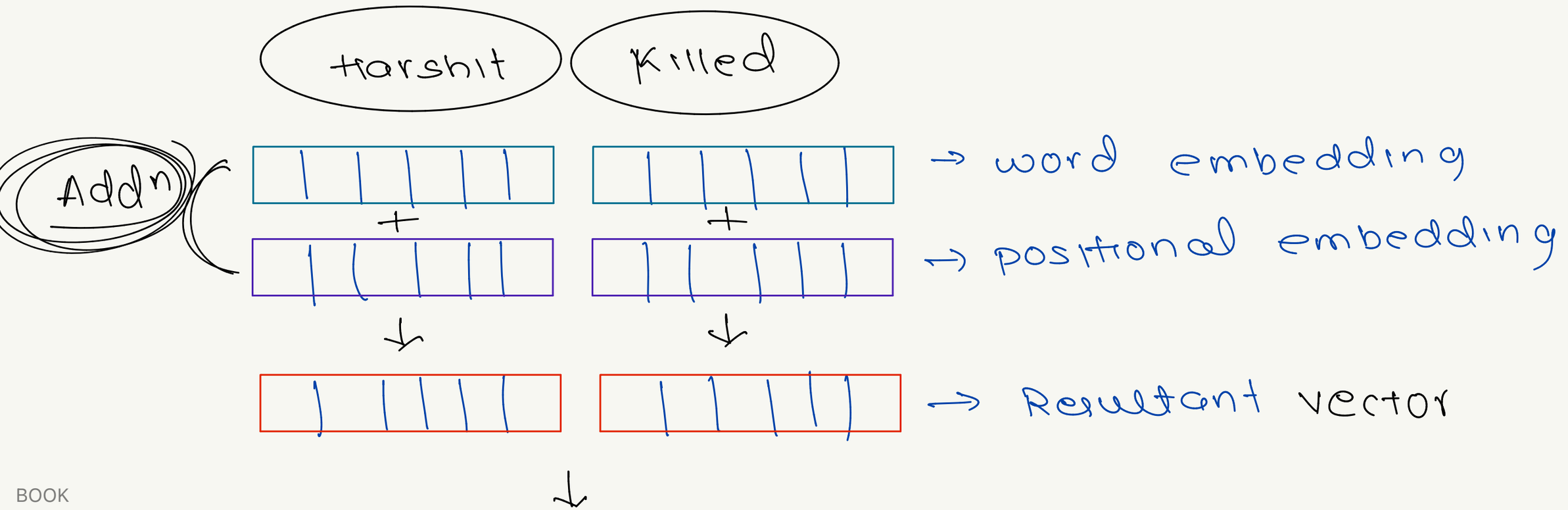


Now probaity of same positional encoding has decreased more.

So, even now we can add more pur cos(π/4) & sin(π/4), sin(π/3) cos(π/3) etc



Har word ke positional embedding vector ka dimen-sion exactly same hoga as dimention of embedding vector.



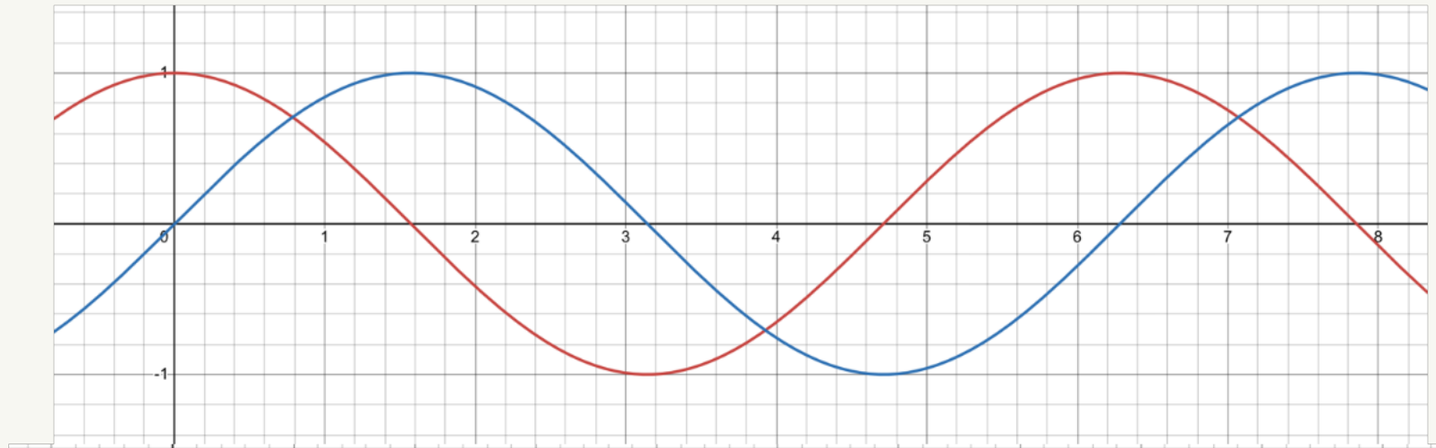
Self Attention

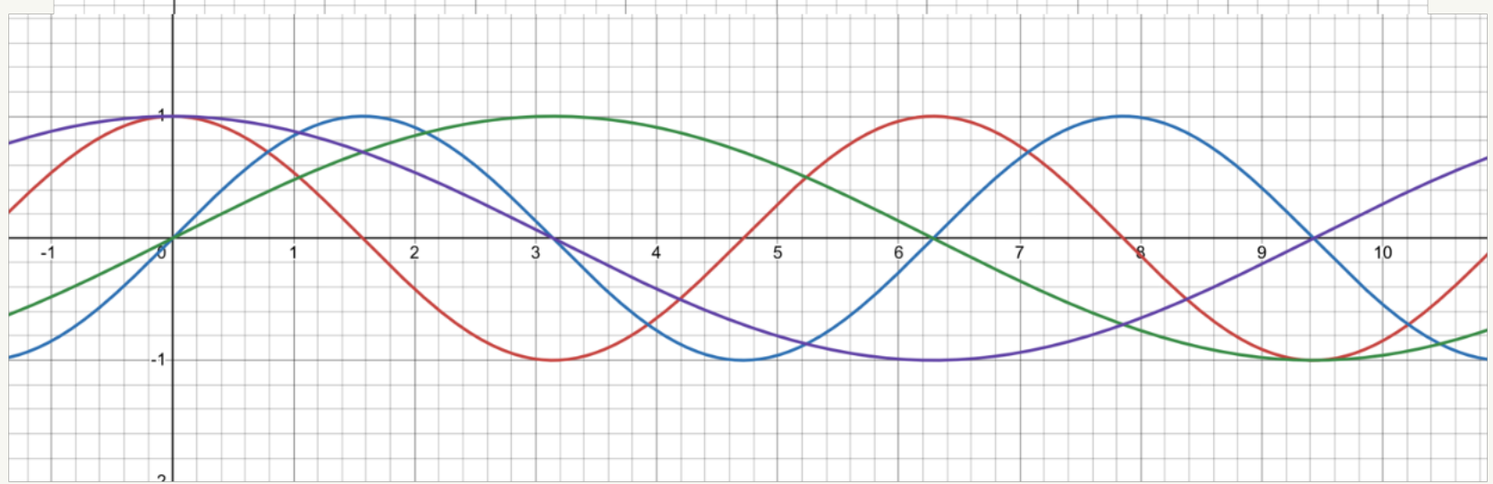
⇒ what is inside positional embedding.

Lets consider if positional vector is of 6 dimention

River	$\sin(1)$	$\cos(1)$	$\sin(1/2)$	$\cos(1/2$	$\sin(1/3)$	$\cos 1/3$
Bank	$\sin(2)$	$\cos(2)$	$\sin(2/2)$	$\cos(2/2)$	$\sin(2/3)$	$\cos 2/3$

Jaise jaise aur sin, cos curves aayenge unki frequency kam hoti jayegi.

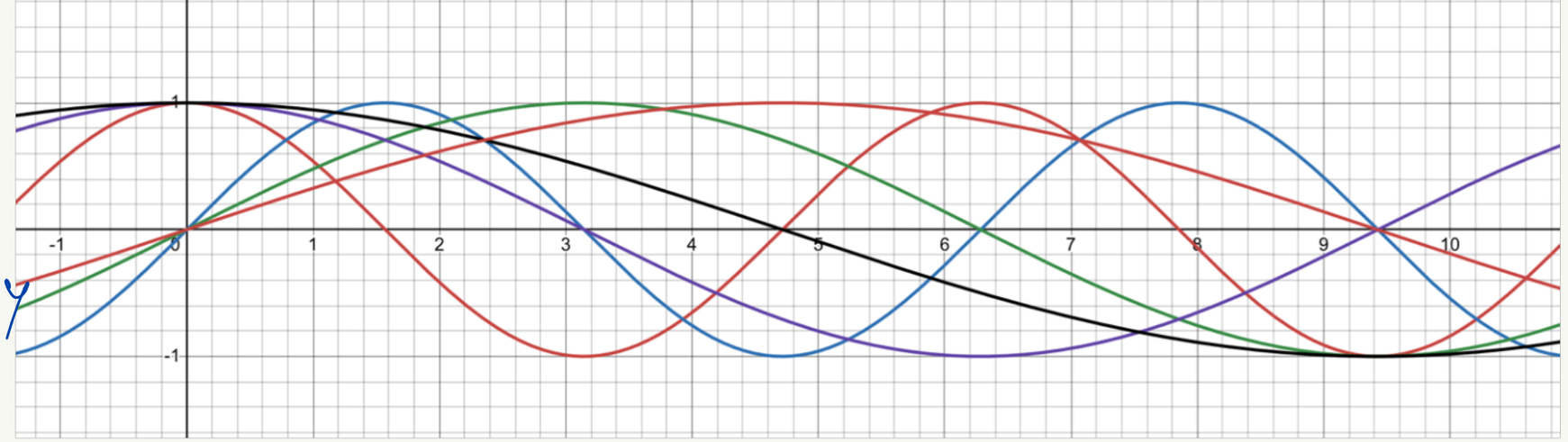




Exact formula

pos → position of word

dmodel → dimentionality of word



$$PE(pos, 2i) = \sin(pos / 10000^{2i/dmodel})$$

$$PE(pos, 2i+1) = \cos(pos / 10000^{2i/dmodel})$$

Example

River

1	2	3	4	5	6

Bank

1	2	3	4	5	6

word embedding → 6D

positional embedding → 6D

pos → 0

dmodel → 6

pos → 1

dmodel → 6

$i \rightarrow 0$ to $\frac{dmodel}{2} - 1$

i.e $0 - 2$

$i = 0, 1, 2$

For River

1D → $PE(0,0) = \sin(0/10000^{0}) = 0$

2D → $PE(0,1) = \cos(0/10000^{0}) = 1$

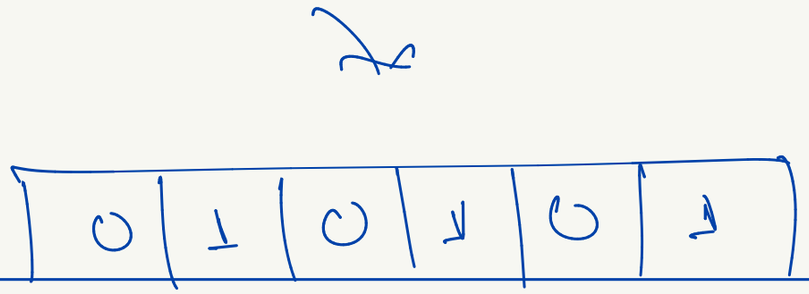
3D → $PE(0,2) = \sin(0/10000^{-1/3}) = 0$

4D → $PE(0,3) = \cos(0/10000^{-1/3}) = 1$

& so on

5D → 0

6D → 1



$i \rightarrow 0$ to $\frac{dmodel}{2} - 1$

i.e $0 - 2$

$i = 0, 1, 2$

For Bank

using same formula -

1D → 0.84

2D → 0.54

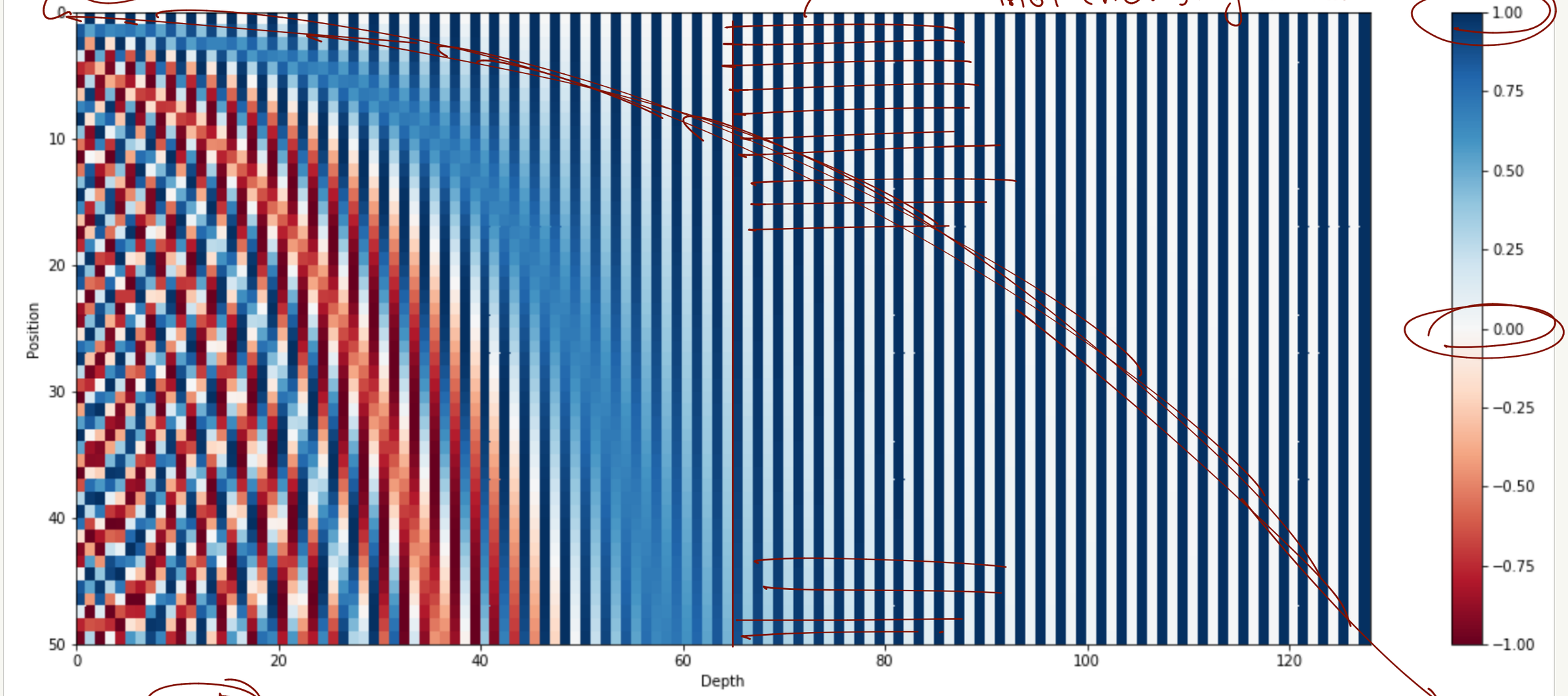
3D → 0.04

4D → 0.99

5D → 0.00

6D → 0.99

This graph shows positional encoding of 50 words each of 128D.



-1st position

↗ higher dimention is not changing much

1st

1010101010101 →

2nd

3rd

0

50th word

50

0 → variations ← 20

40

60

Depth

white → 0

Blue → 1

Agar no of words jyada ho to higher dimention men bhi change acyega

ChatGPT

Masked Language Modeling (MLM) is a type of pre-training task used in training transformer-based language models, most notably BERT (Bidirectional Encoder Representations from Transformers). MLM is crucial for enabling these models to understand and generate human language in a context-aware manner. Here's a detailed look at MLM:

Key Concepts

1. **Masked Tokens**:
   - During training, a percentage of the tokens in the input sequence are randomly replaced with a special `[MASK]` token.
   - Typically, 15% of the tokens are masked in the training sequence.
2. **Contextual Learning**:
   - The model is trained to predict the original tokens that have been replaced by the `[MASK]` tokens.
   - This requires the model to use the context provided by the surrounding (unmasked) tokens to infer the masked token.