Seq2Seq Data

Campus X

1) Tabular Data

Cgpa, IQ, placement

train it on ANN → add more layers → DNN
(artificial) (Deep)

improve

2) Image Data

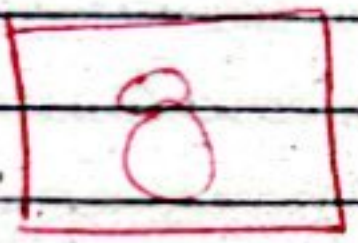


image classification

↓

is made of pixel

↓

2D structure

↓

has some meaning that can't understand ANN

↓

then

↓

CNN

3) Sequential Data

→ textual Data

→ time series based data.

sequence (order) has some important but unfortunately ANN and CNN can't understand

⇒ RNN

LSTM GRUs

4) Seq2Seq Data :-

Such data where input is also sequence and output is also sequence
e.g machine translation

{ input seq → output seq } ⇒ machine translation

. nice to meet you (4 words)
⇓
(2) (3) آپ سے مل کر خوشی ہوئی (6 words)

we made such neural n/w architecture that handle seq2seq data.

Q Why it is difficult to solve seq2seq Data?

1) input → sentence in some language
↳ variable length.

2) output → sentence in some language
↳ variable length

3) not guarantee input & output seq2seq Data of same length

it mean with dealing with variable length is difficult

input → output
6 word 600 words

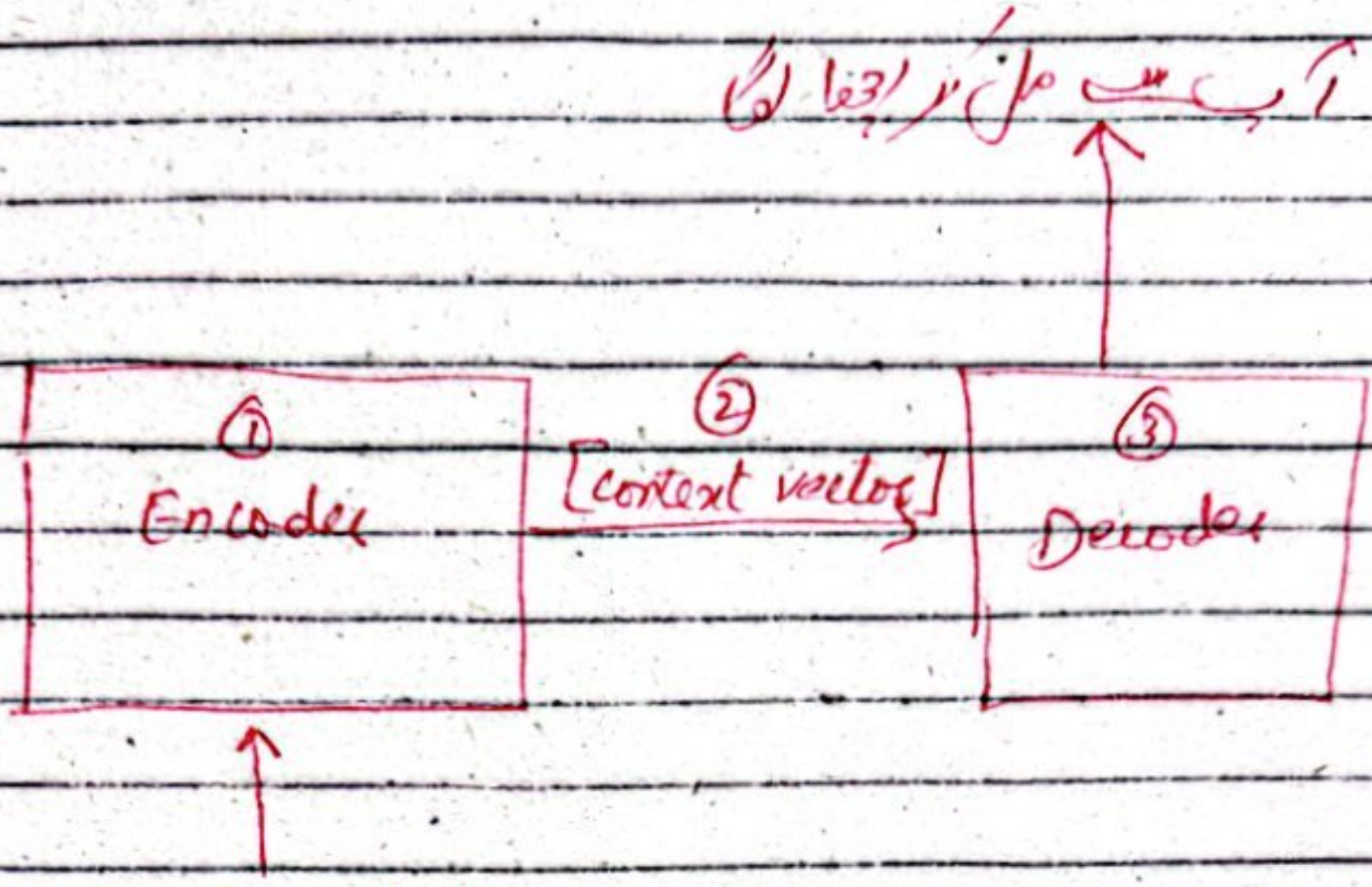
variable length
↳ handle we learn in LSTM & GRU but only for input

but not for output and that we learn to solve seq to seq problem.

Encoder Decoder Architecture

Pre-Requisit
→ RNN / LSTM

→ Sequence to Sequence with Neural N/w research paper



Encoder :-

is basically that section of architecture where input sequence give.

↓ to make sure input given token by token (word by word)

Encoder → to understand complete sentence
↳ to capture its essence
↳ read by to summarize

↓ output

vector (set of numbers) that called (context vector.)

Context vector is basically summary of this sentence.

That made from encoder intelligent

that context vector give to (decoder)

to understand context vector (summary)

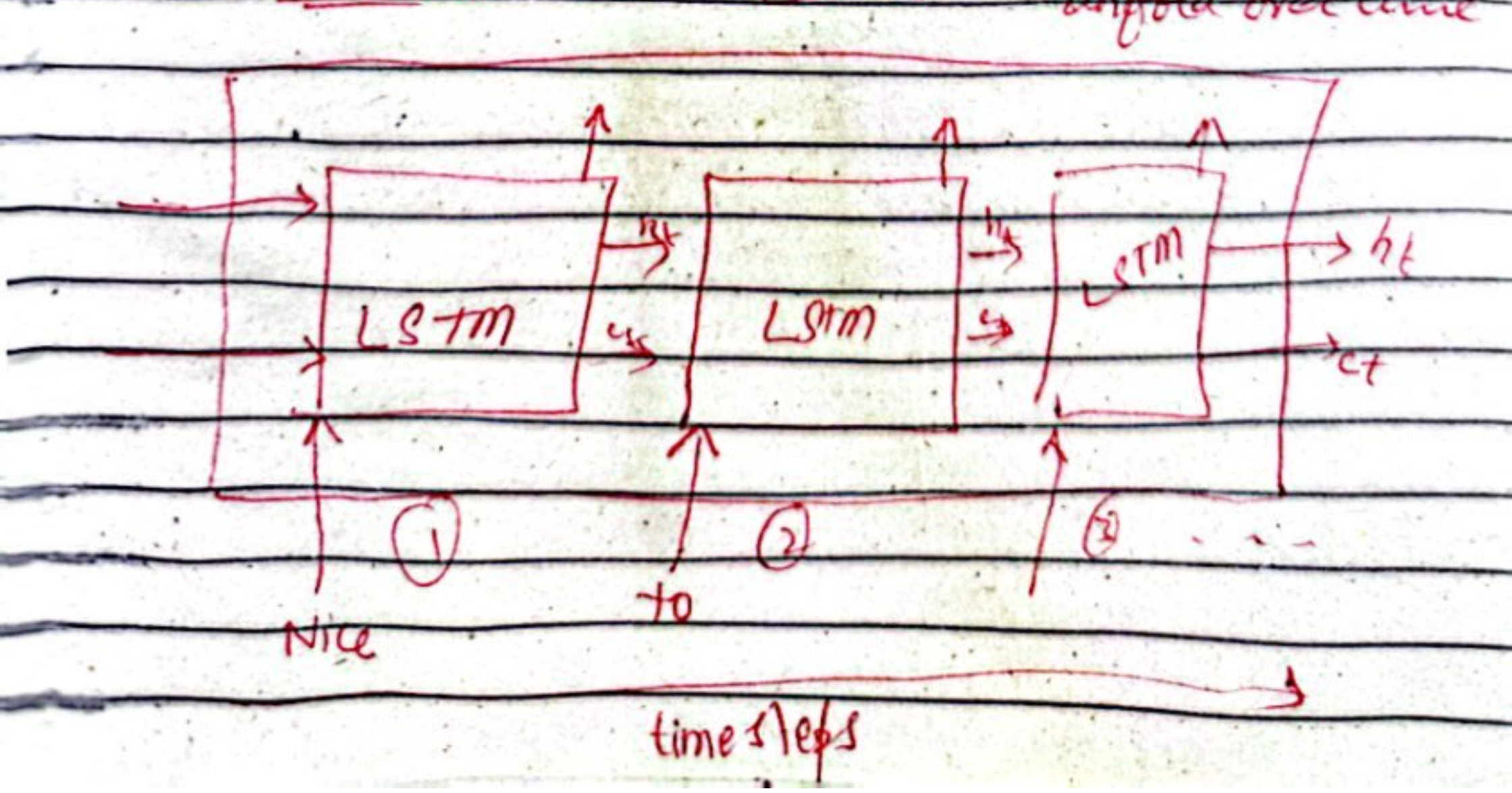
and then token by token (word to word) output.

What's under the encoder and decoder

↳ we want such thing - that process sequence in both encoder & decoder.

Encoder ⟹ 

Encoder is one LSTM cell, we unfold over time



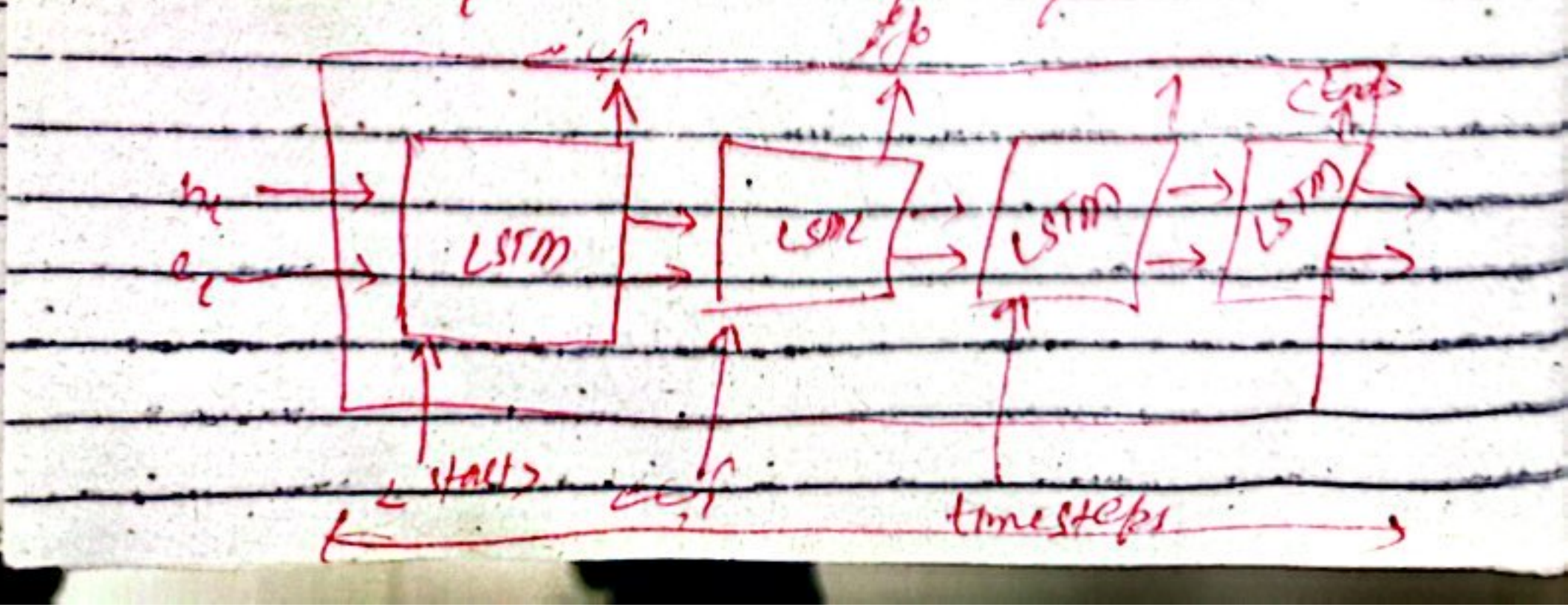
In whole process LSTM understand sentence and send its summarization through hidden layers.

After final step, we get h_t and c_t that is our final representation of sentence that become the Context vector that h_t and c_t pass to the decoder.

In summary Encoder is made of one LSTM
it is not necessary to used LSTM you also used GRUs

RNN generally not used because it has vanishing problem.

Decoder :-
also has one LSTM
⇒ here LSTM works is in every time step some output produced.

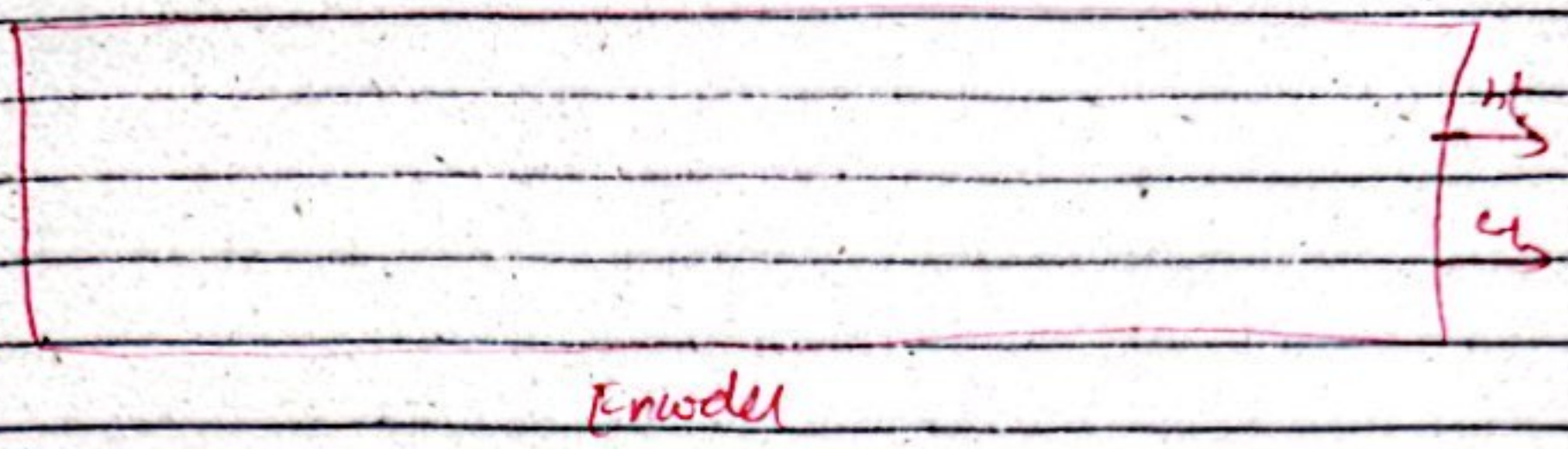


h_t and c_t that we got from encoder that pass it to decoder through initial state.
and in that way decoder get content of input sequence.

at timestep 1 you not only pass content vector but also pass special symbol as input which we call <start>.
By seeing <start> decoder understand now it start produce output.
and it produce any one word output At timestep 1 and that output we pass at input at timestep 2 along with internal state and so on until output <END> token ~~find~~ get.
And it stop output produced.

Training

Dataset — machine translation
English-Hindi_dataset (Kaggle)



English	Urdu
1) think about it	سوچ کے
2) come in	اندر آ جاؤ

first convert in number.

→ first convert into vocab

[think, about, it] → [سوچ, کے]

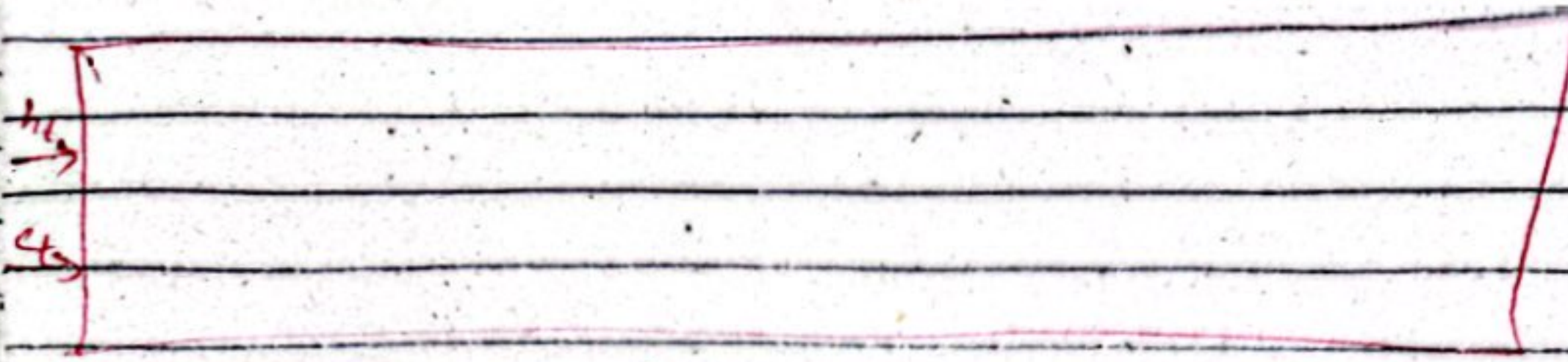
[come, in] → [اندر, آ, جاؤ]

Now convert into number

→ one hot encoding

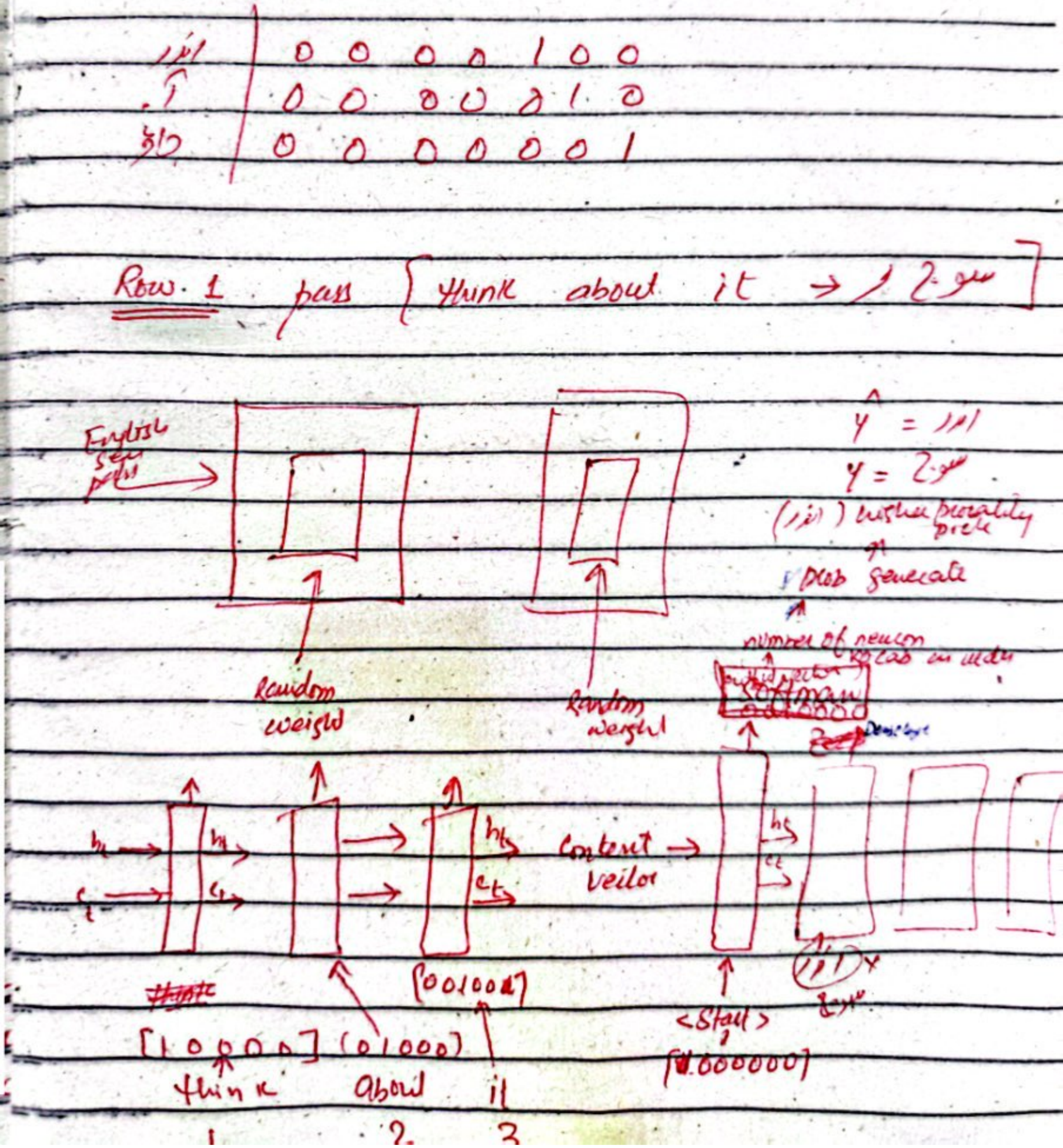
eng	one hot encoding				
think	1	0	0	0	0
about	0	1	0	0	0
it	0	0	1	0	0
come	0	0	0	1	0
in	0	0	0	0	1

Urdu	
<start>	1 0 0 0 0 0 0
<end>	0 1 0 0 0 0 0
سوچ	0 0 1 0 0 0 0
کے	0 0 0 1 0 0 0



پدر	0	0	0	0	1	0	0	
آ	0	0	0	0	1	0	0	
کتاب	0	0	0	0	0	0	1	

Row. 1 pass [think about it → آ پدر]



in decoder.

→ pass context vector and <start> symbol as input.

→ it generate some output, softmax layer. (output probs)

→ suppose it generate 'i'

→ ~~But~~ we pass that output as input but it is incorrect.

So researcher think if we pass incorrect as input (Training become worst)

So it decide

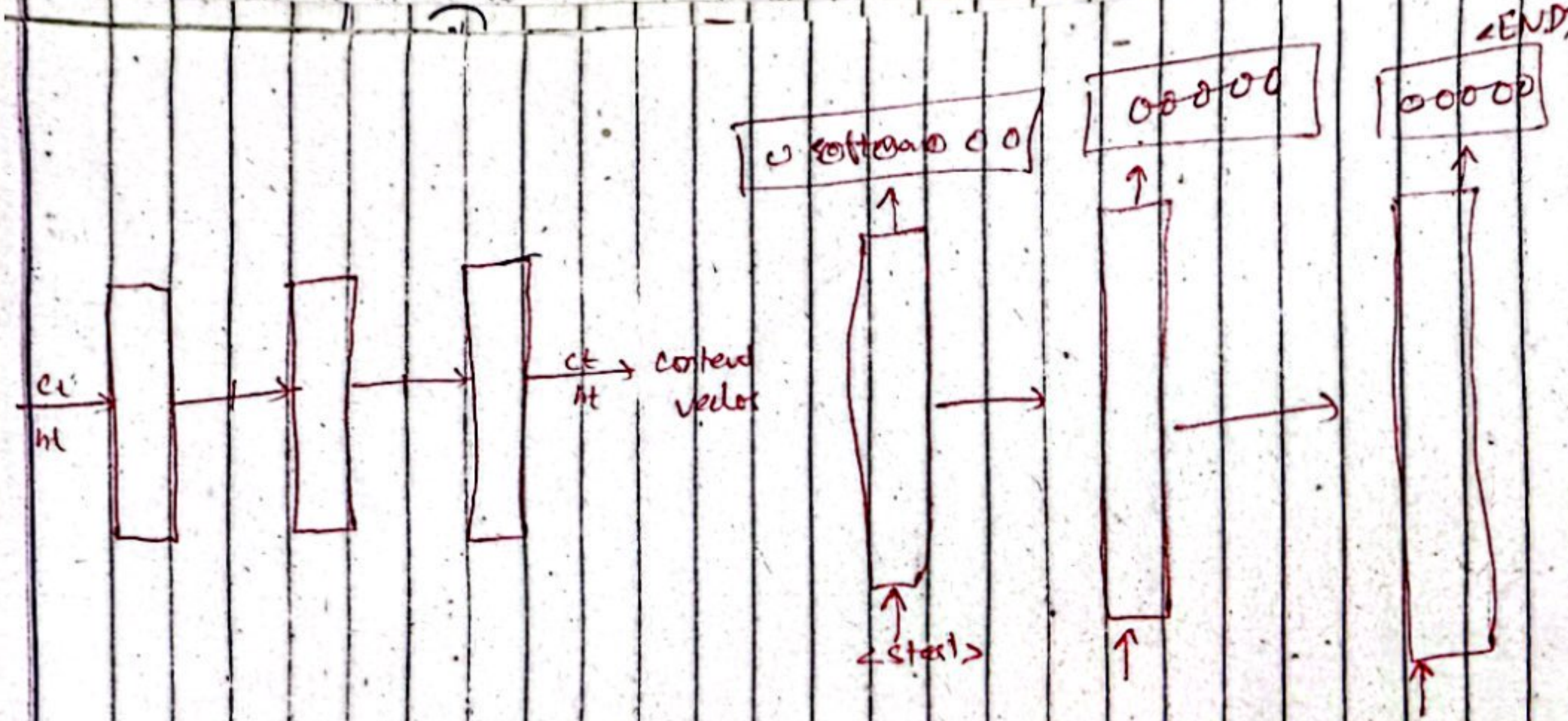
whether it got incorrect output. it give correct word as input

That's is called Teacher Enforcing

it benefit is training fast convergence

→ total loss

→ average loss



y_actual: [0 0 1 0 0 0 0] <END>

$\hat{y}$_predicted: [0.2, 0.01, 0.01, 0.7, 0.01, 0.1, 0.5] <END>

t=1   t=2   t=3

Loss — Category loss entropy

$$L = -\sum_{i=1}^{7} y_i \log(\hat{y}_i^{pred})$$

$L_1 = -1 \log(0.01)$ $L_2 = 1$ $L_3 = -1 \log(0.04)$ $= 0.39$

total loss = 2.37

avg loss = 0.7

initial weight → forward pass → output → compare with correct output → loss → Gradient → used → weight update using some optimizer ADAM → new weight → learning rate → which rate (speed)

1) forward
2) Loss
3) Backpropagation
4) gradient
5) update parameter

→ measure any one perturbe parameter how it contribute in loss function and that parameter adjust in which direction where loss less.

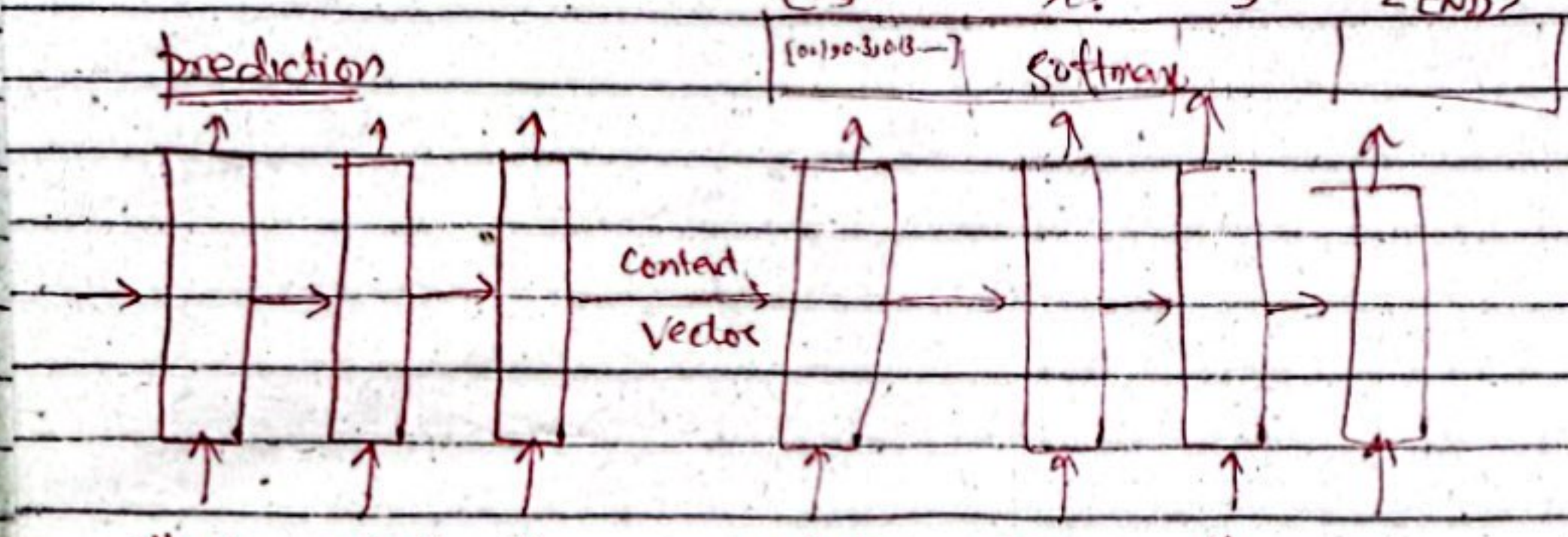
→ Stocastic GD, ADAM

Optimizer adjust weight in that direction where loss less and that adjustment we based on learning rate hyperparameter & Gradient

Prediction

① ② ③ ④ ⑤ ⑥ ⑦

<start> سوج لو انور آ جاؤ <END>



think about it <start> آ جاؤ لو

[1 0 0 0 0 0 0] [0 1 0 0 0 0 0] [0 0 0 0 0 0 1] [0 0 0 0 1 0 0]

input → [think about it] } ×

آ جاؤ لو

Improvement 1 [Embedding]

one hot encoding

⇓

if 1 lac it means 1 - lac data

it does not need

⇓

~~low~~ Embedding used

⇓

low dimension of encoding

⇓

~~dense~~ Representation

↓

suppose 30 → summary of word

⇓

that can't get it

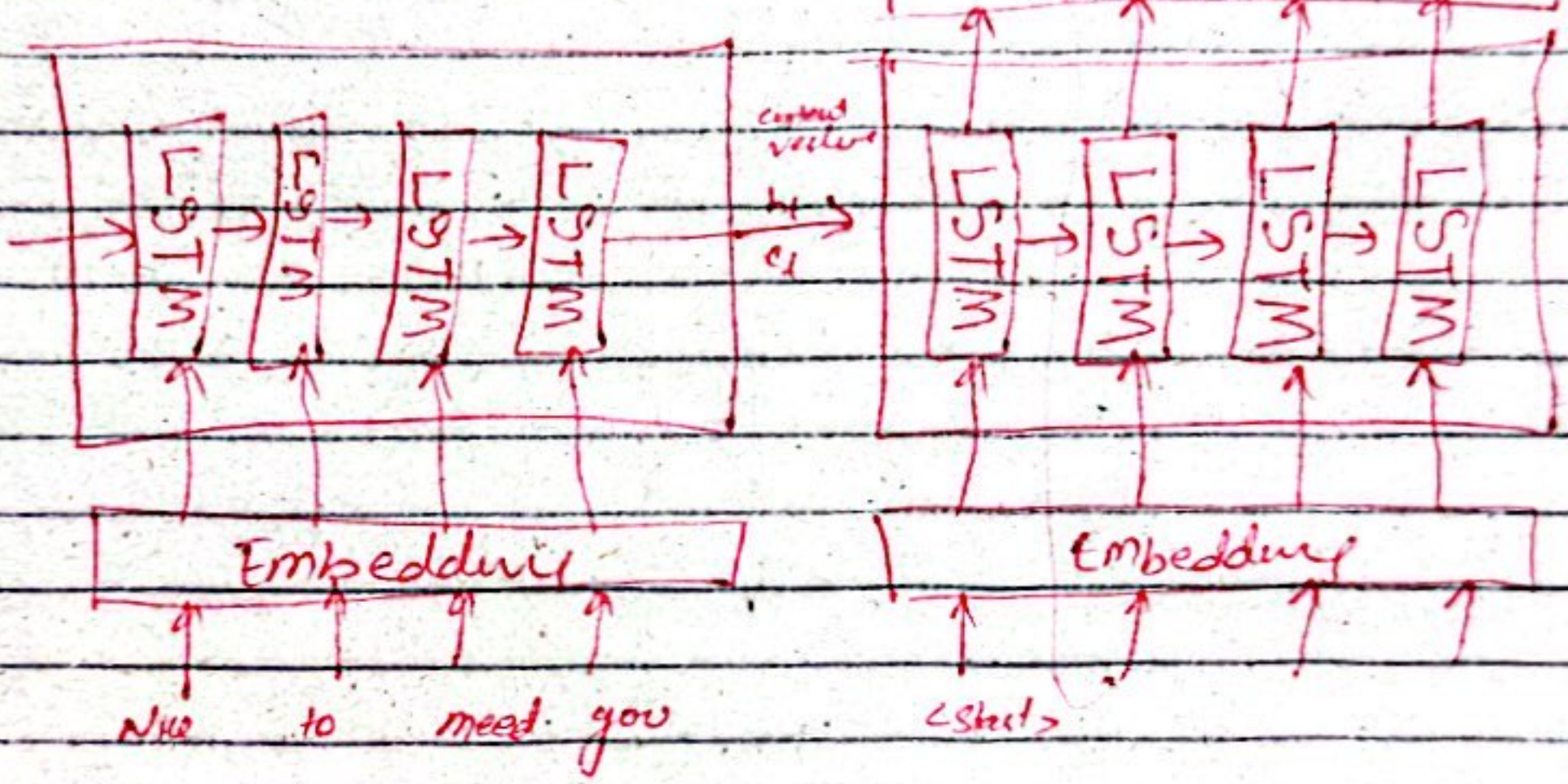
one hot encoding

Embedding

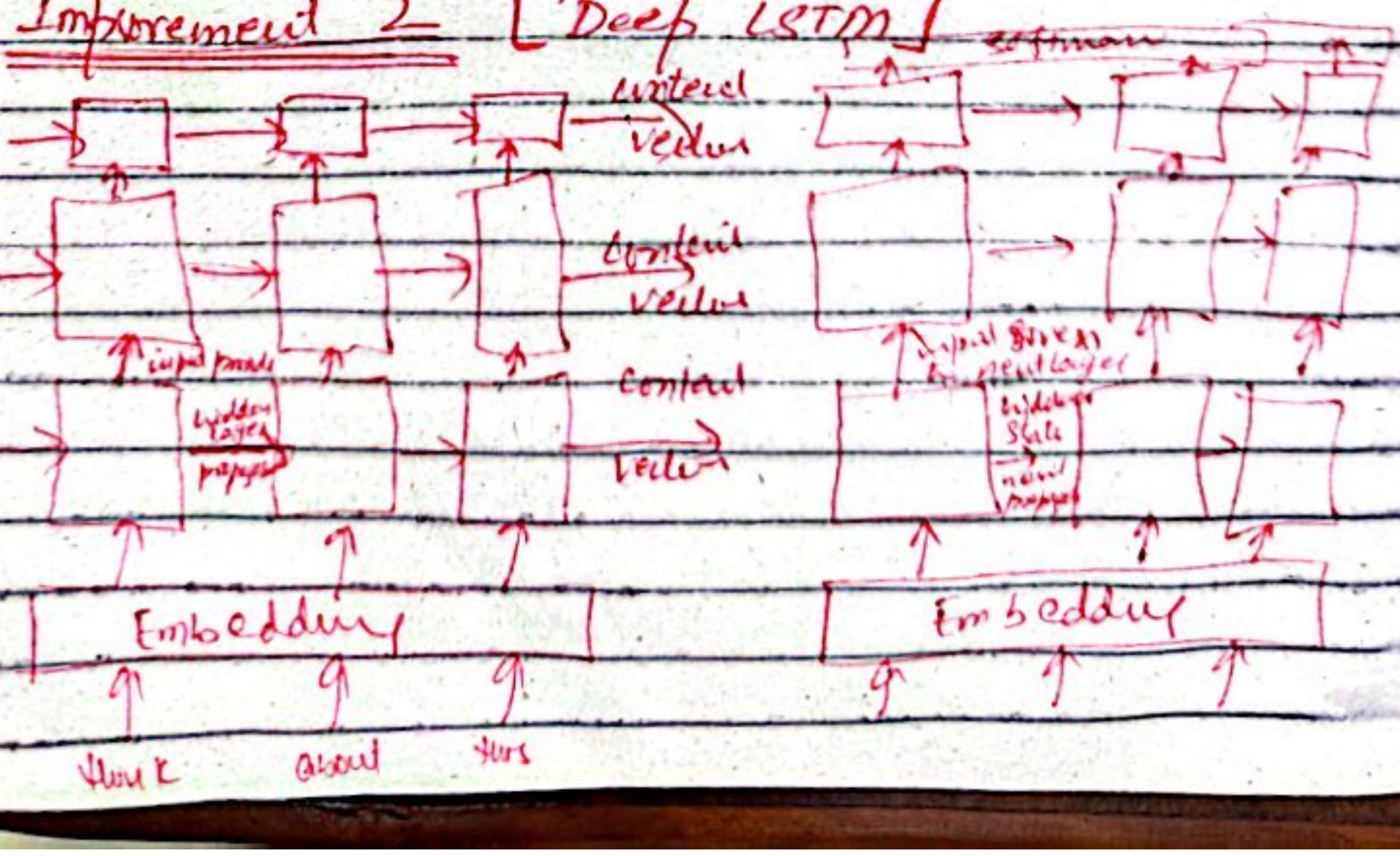
pretrained word2vec / Glove

train (own dataset corpus)

Encoder & Decoder both used Embedding



Improvement 2 ['Deep LSTM]



3 Reason why you do this.

1) Long term dependencies
more efficiently handle
(Long paragraph)

Single layer performance in long paragraph
is not good.

In deep Encoding and Decoding
1) context vector get from multiple layer
we have more capability to store/
represent that summary

If one context vector is not efficiently
give summary of single sentence

2) Layered representation :-

Hierarchical data more understand
LSTM
→ lower level understand
(word level) → singular plural
→ verb
→ pronoun

→ middle level LSTM
(sentence level) understand
↳ context understand

→ top level LSTM
paragraph level understand
↳ full paragraph content.

eg The phone battery is bad but the overall
phone is good

(3) Deep learning NN

Whenever you increase DNN parameters, its learning capability increase mean it capture minute until not overfit

capacity increase ⇒ small thing more capture

→ better generation
if sufficient data provide not overfit and new data it better give results.

~~not~~ t

→ input variable also better capture

Improvement 3

Reversing the input

benefit

start words is more closer and it pick good content gradient backpropagate less effort.

Drawbacks

last words more far

It work on such language

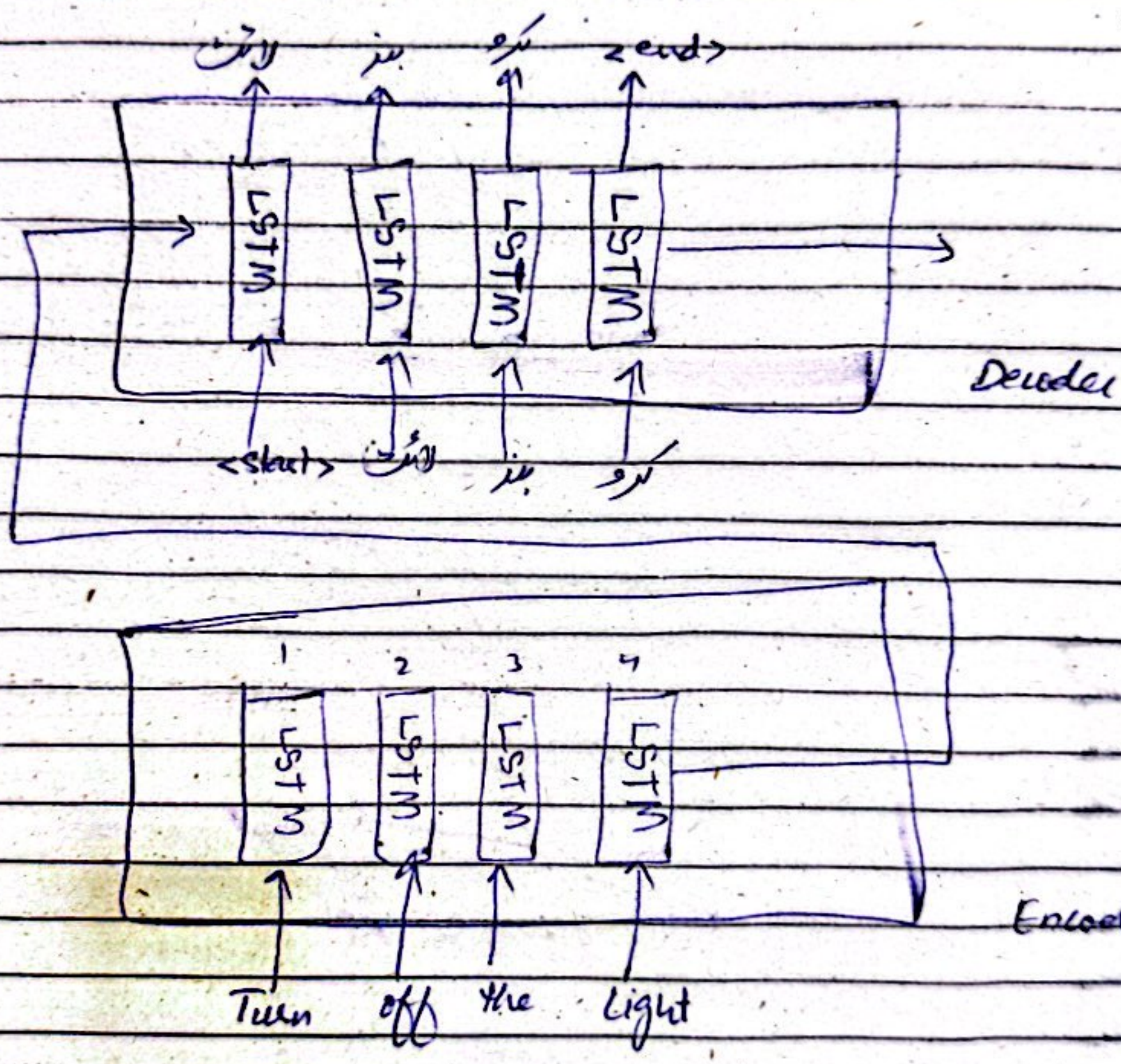
→ initial content (English → French)
more information

but not for many languages.

Attention

Why we need attention?

Encoder Decoder Architecture



you provide one sentence to encoder. Encoder step by step process.
one time it process whole then it generate summary & context vector & and pass that vector as input in decoder.
decoder step by step output sentence

problem of Encoder & Decoder :-

So word sentence, and one time summarize and then translate is very difficult.

problem is word > 25 content vector load is ~~too~~ very huge.
very small set of numbers, you give very huge responsibility.

↳ unnecessary pressure create on Encoder
⇒ at a time whole sentence read and learn and tell it decoder

↳ at any time step at decoder, we don't need input sentence to translate we need some specific word / set of words

but problem in this architecture. whole input sentence give at very time stamp : go and pause out.
which word you need to give decision

⇒ because of static representation, decoder face problem to decode

best ~~@~~ dynamic if it is, any particular output point only specific part attention

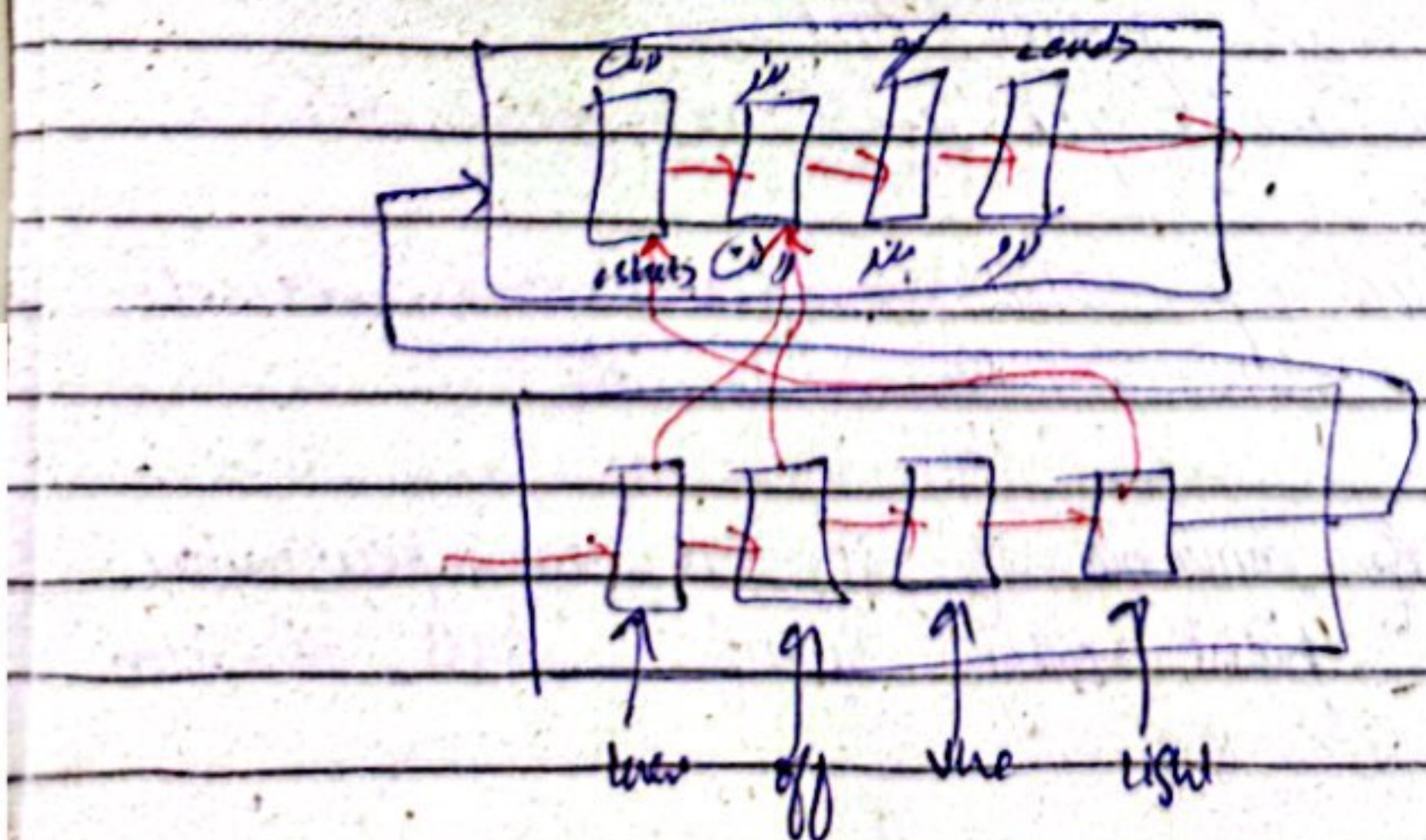
Solution

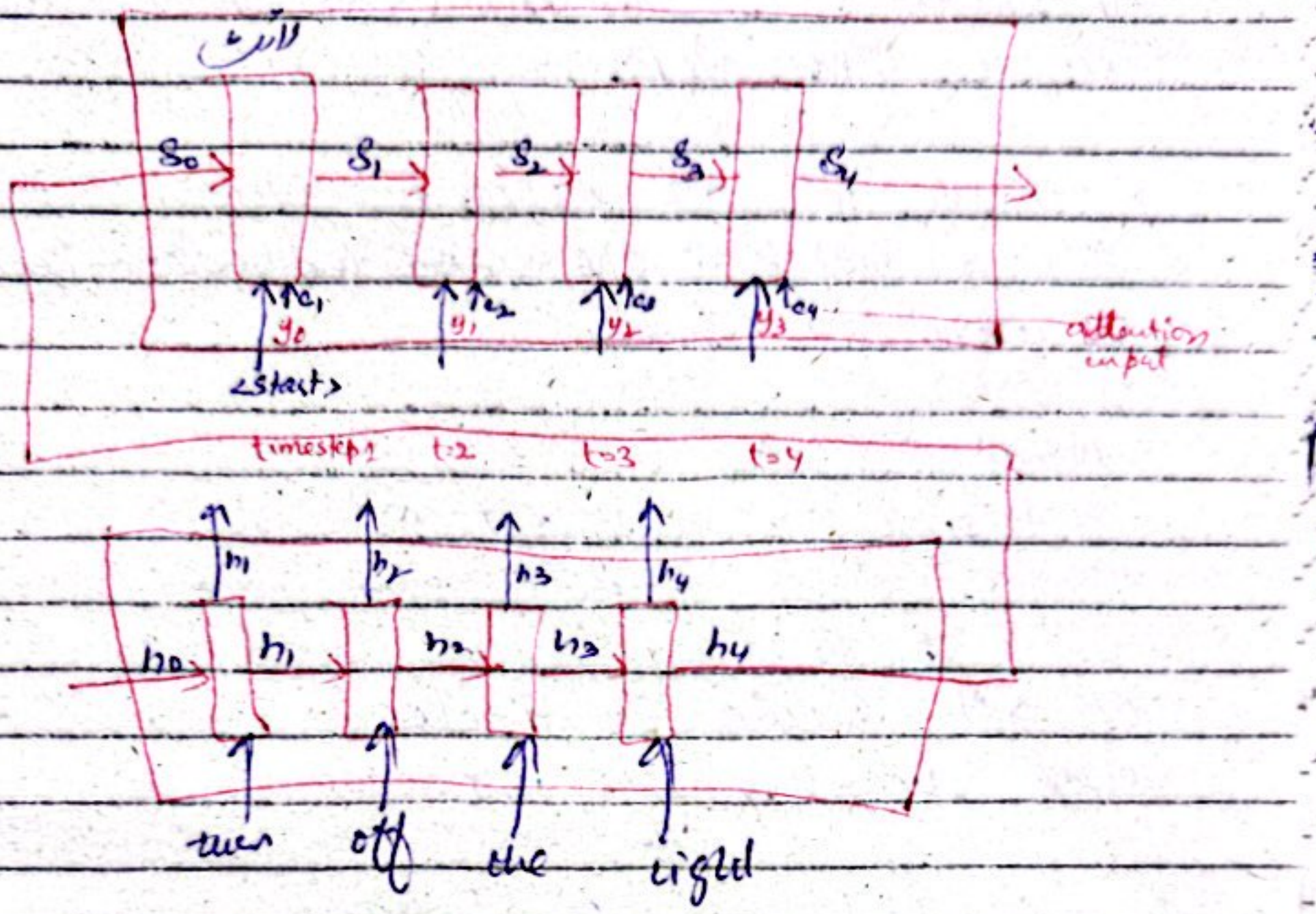
human being if translate long sentence it only focus particular part of sentence and translate it. other part is blur.

This concept we introduce in our architecture. In decode at timestamp1 to print ~~decoder~~ decoder output. we tell information in any way of Decoder that encoder timestep is very important to print 'a' and to print in decoder timestep 1 and 2 are important and so on.

we dynamically information generate at every time stamp of ~~every~~ decoder, which encode timestamp or set of time step important accurately. and that mechanism is called

Attention mechanism.





we know that, in attention mechanism we pass out information which encoder time step is important for decoder each time step.

Decoder $t=2$ (y_{t-1}, S_{t-1}) provide input (vanilla)

output hidden

In attention you also pass

$t=2$ y_{t-1} S_{t-1} c_t

context vector input

its purpose to tell decoder at current timestep which encoder hidden state is useful

what c_i is look like?

1- vector
2- scalar
3- matrix

} to answer that question first look what is the purpose of c_i?

Decoder first time step which hidden state of encoder it accept
by hidden state is useful to print it all
decoder first step

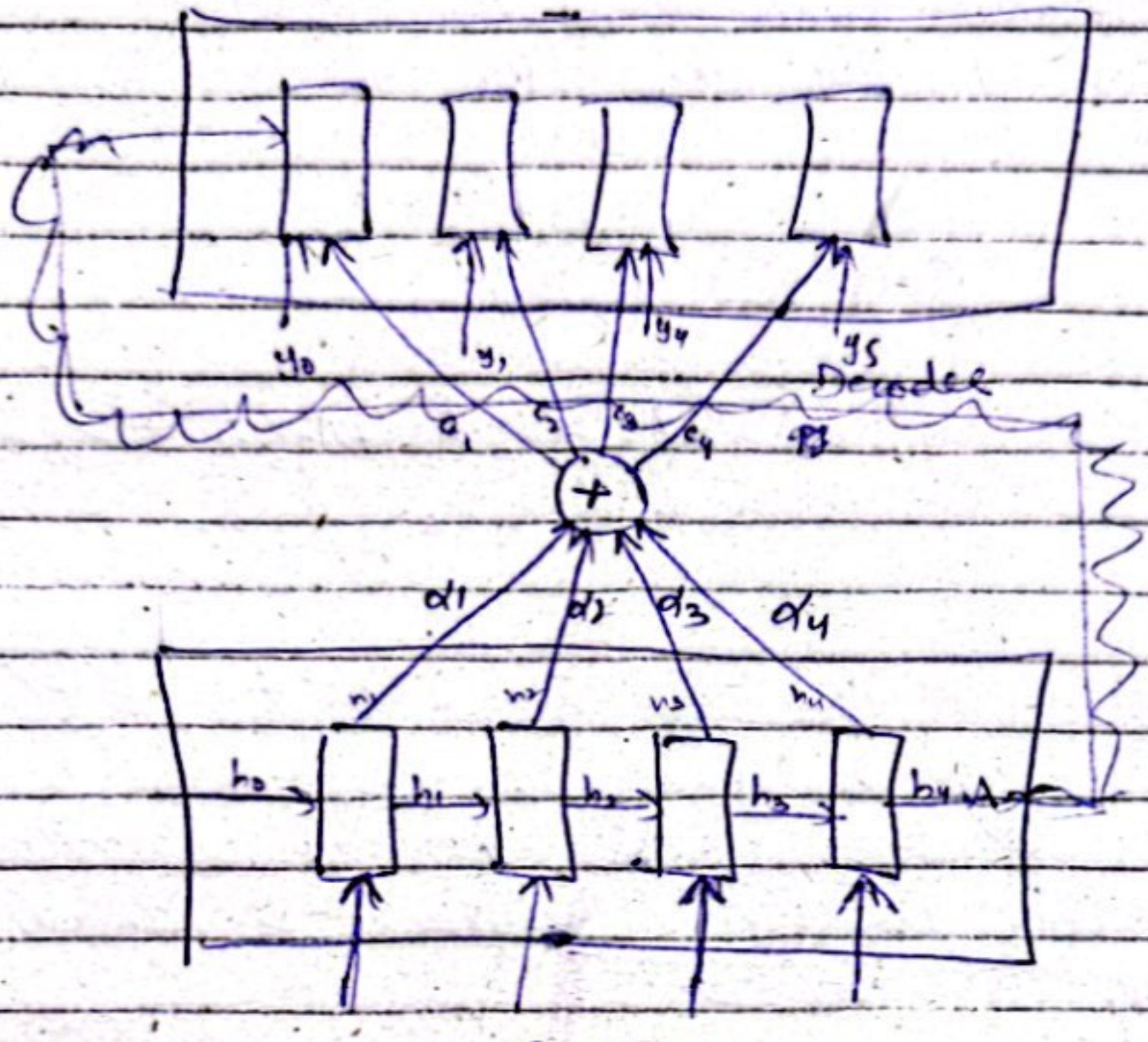
if someone tell me magically which hidden
state is important for decoder each time step
c_i accept it only $h_4 \Rightarrow c_i$

Decoder
$t=2$ h_1 and h_2
$t=1$ h_4

c_i ⇒ It should be vector
which dimension of c_i.

[1 0.5 0.6 0.3] ⇒ h_1
⇒ h_2
⇒ h_3.

c_i exactly same as h_i.
but what if, I will send two hidden
we weighted sum. and send it.



Decoder time step "i"

Encoder time step "j"

In attention mechanism:-

it assign weight of all hidden state h_j

$$c_{i4} = \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3 + \alpha_4 h_4$$

scalar

vector

dimension

c_4 = 4 dimension

If we find c_4, we find weight for each hidden state, which α_i is dominate its input is highest.

for each time step ... for decoder
α_i is different for each time step

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

$$c_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3 + \alpha_{14} h_4$$
$$c_2 = \alpha_{21} h_1 + \alpha_{21} h_2 + \alpha_{23} h_4 + \alpha_{24} h_5$$

total α = $i + j$ = $4 \times 4 = 16$

How α are create? how it calculate, because it has informations which hidden state is important.

α_{21} → alignment / similarity score which tell decoder 2nd time step which output is point, it output encoder last time step how much role.

α_{21} (2nd time step) — encoder last step was more important

α_{21} which quantities it depend?

h_1 S_1 → previous hidden state of decoder how?

given translation based second step output how much role?

$$\alpha_{21} \rightarrow f(h_1, S_1)$$

$$\alpha_{ij} = f(h_j, s_{i-1})$$

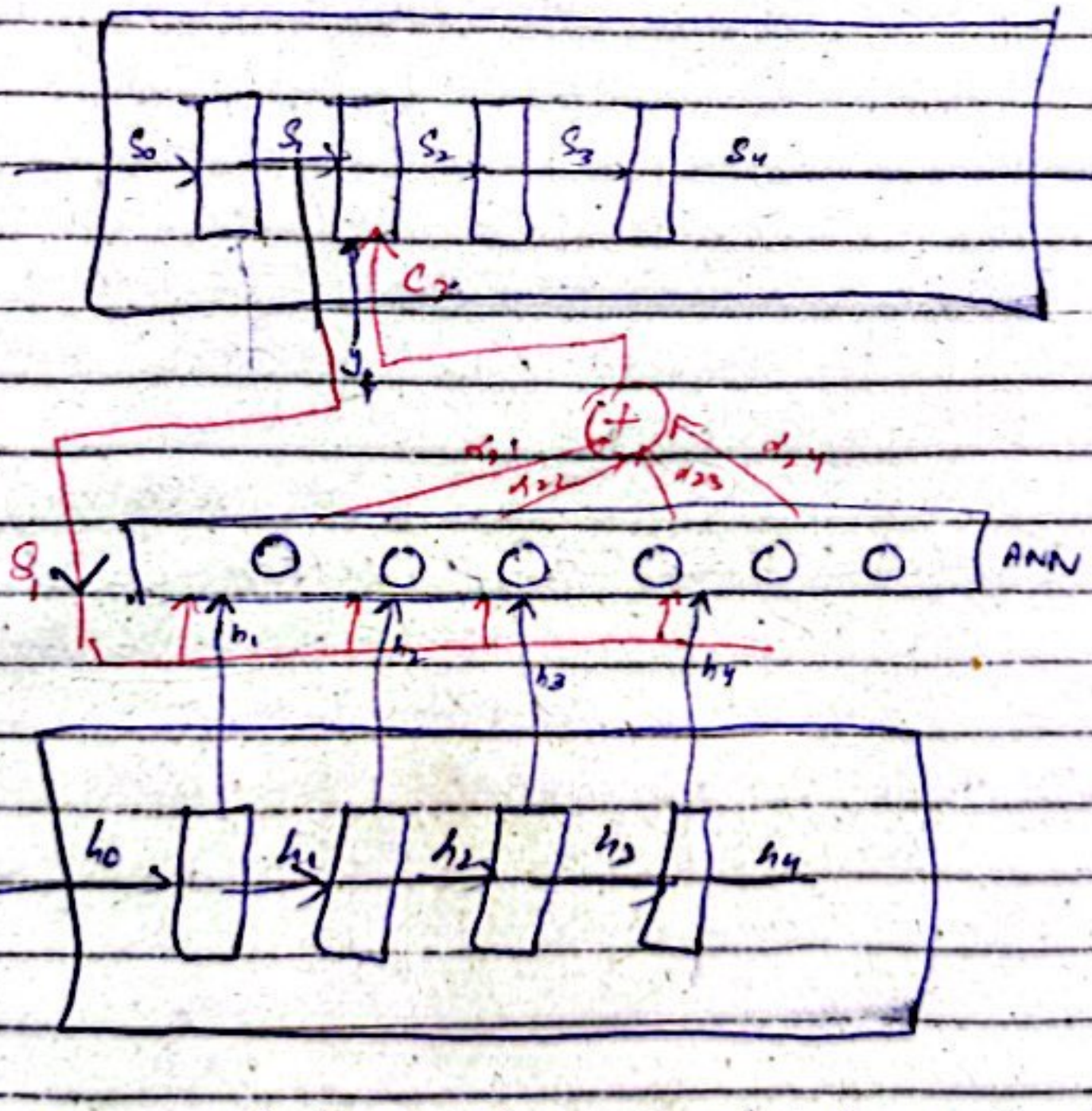
$$\alpha_{23} = f(h_3, s_2)$$

what function is used?

researches - think why it think function. :)

ANN → universal function Approximator.

we can directly used neural n/w.



$$c_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3 + \alpha_{24} h_4$$

BLEU Score ⇒ translate Quality

Sentence Length > 30 BLEU score maintain/stable in attention mechanism.

And other BLEU score > 30 decrease

attention head = α ⇒ plot.

	Head 1	Head 2	Head 3	Head 4
start	α_{11}	α_{12}	α_{13}	α_{14}
je	α_{21}	α_{22}	α_{23}	α_{24}
suis	α_{31}	α_{32}	α_{33}	α_{34}
end	α_{41}	α_{42}	α_{43}	α_{44}

English to French translation. (paper)

its tell benefit of attention

encoder ⇒ bi-direction LSTM
backward and forward context

decoder ⇒ LSTM

Bahdanau Attention vs Luong Attention

Machine translation :- NMT (Neural Machine translation)
turn off the light ... لائٹ بند کرو

Encoder & Decoder

→ LSTM, bi-LSTM, stacked LSTM used as your wish.

Step by step each word sentence, encoder goal is to generate summary context vector) and that context vector passed to decoder.
Decoder work it step by step try to print output.

problem is

Sen > 30 paragraph, document encoder face difficulty to generate content vector (set of numbers) how much data you ~~create~~ store. Context vector can't create efficiently and that's why decoder can't generate good quality translation.

To eliminate that problem new mechanism came that is Attention.

Decoder any particular timestep, you point a word, you don't need full sentence, you need words or subset of words.

light
turn off.

بتر

Attention mechanism said, don't need of send whole content vector.

we dynamically select h_i.
at each time step of decoder, content vector send. c_1, c_2, c_3, c_4

اول point $\{h_4\}$ word
یا point $\{h_1, h_2\}$

*

if we count any particular $c_i = \sum_j \alpha_{ij} h_j$

$$c_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3 + \alpha_{14} h_4$$
$$c_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3 + \alpha_{24} h_4$$

Total alphas $\{$ input sentence word x, output sentence word $\}$

alignment score

how we find α (word to word similarity score)
↓ values

α_{ij} =

1 — Bahadonov attention
2 — Luong attention

Bahadonov attention
(Additive Attention)

α = (alignment score) how much importance of decoder time step encoder which time step how much importante

$$\alpha_{ij} = f(h_{fo}, \underbrace{s_{i-1}}_{\text{previous decoder time step}})$$

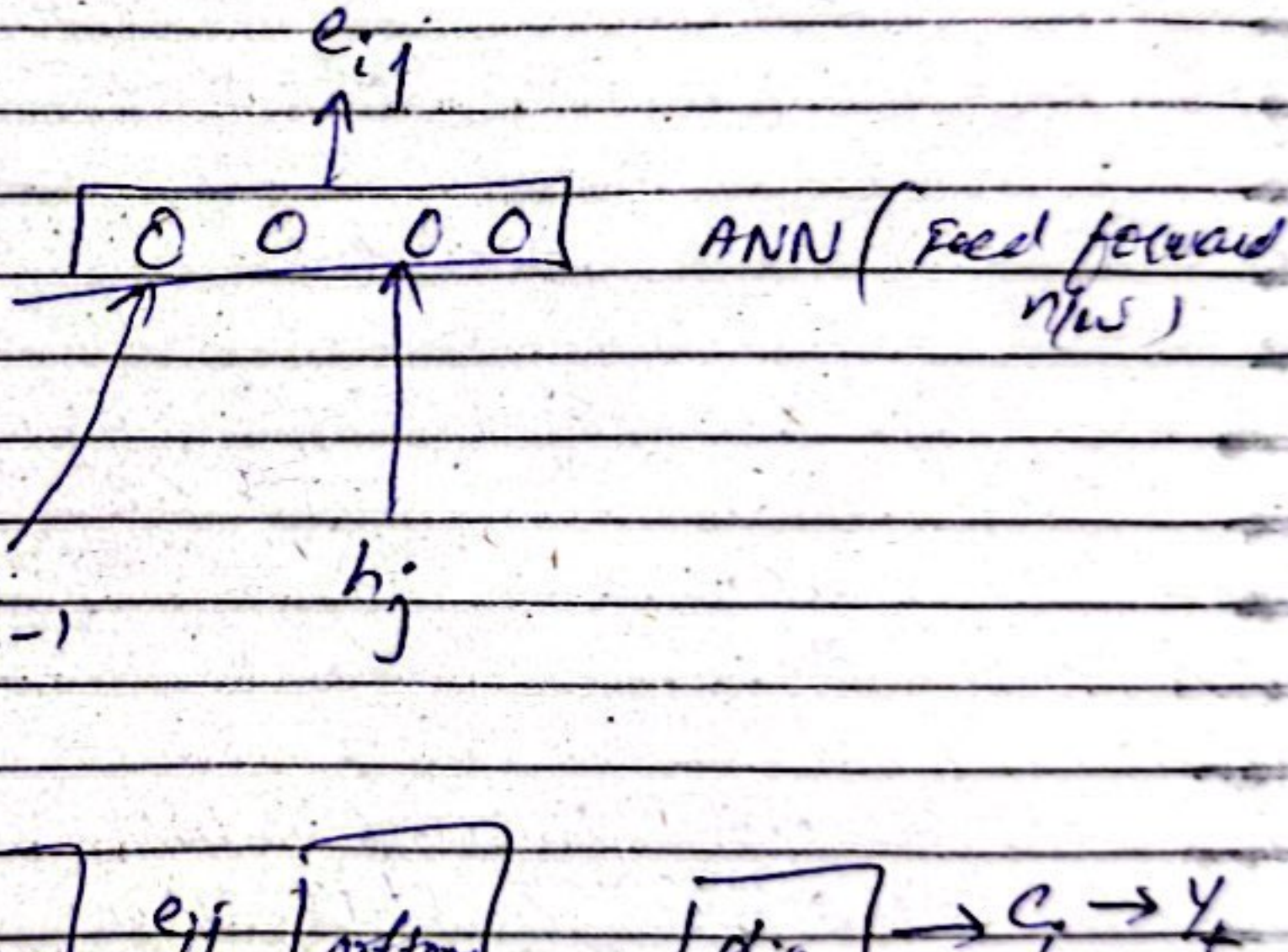
α tell given translation current decoder output how much it depend on encoders which time step.

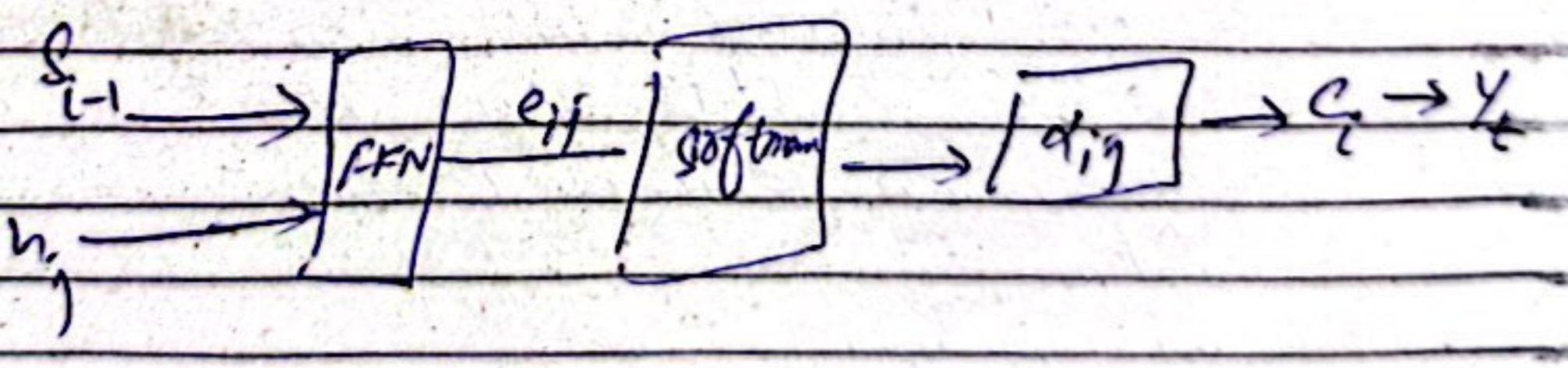
alignment score will be mathematic function of two things
1 — decoder hidden state
2 — decoder previous hidden state

Q which mathematic we used?
A Bahadonov Attention said, why we word hard to find, Wai function

instead we approximate that functions with the help of ANN.

ANN ⇒ approximation function.



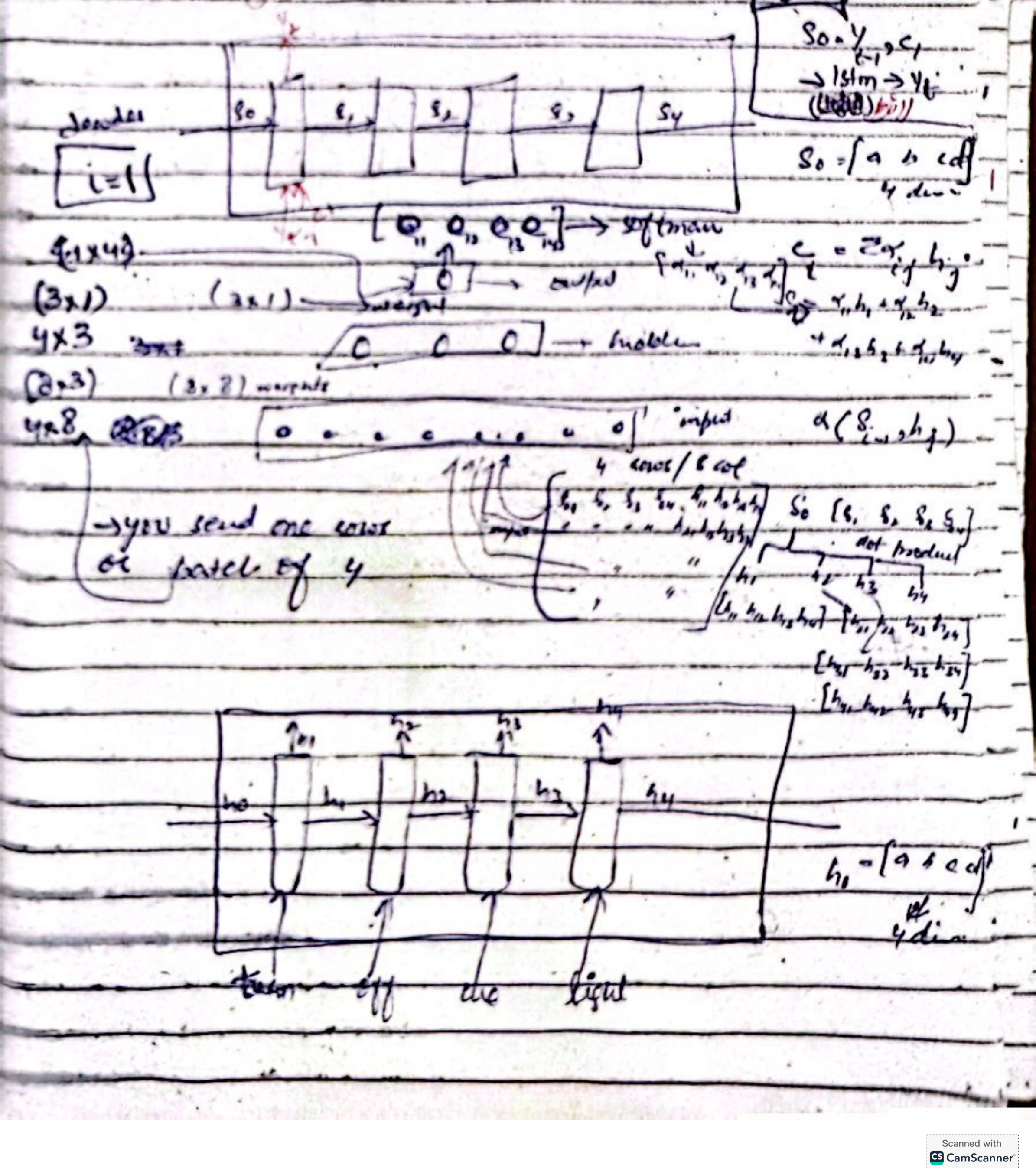


Architecture

Encoder is same, but in decoder you add F-FNN (Feed forward neural N/w)

hidden layer | 0 0 0 | three neurons

1 output



↳ encoder sentence send

h_0, h_1, h_2, h_3, h_4 same

↳ Decoder $t = 1$

S_0 $[S_{01}\ S_{02}\ S_{03}\ S_{04}]$

h_1 h_2 h_3 h_4

$[h_{11}\ h_{12}\ h_{13}\ h_{14}]$ $[h_{21}\ h_{22}\ h_{23}\ h_{24}]$

$[h_{31}\ h_{32}\ h_{33}\ h_{34}]$

$[h_{41}\ h_{42}\ h_{43}\ h_{44}]$

$\alpha_{11}\ \alpha_{12}\ \alpha_{13}\ \alpha_{14}$

softmax

$[e_{11}\ e_{12}\ e_{13}\ e_{14}]$

4x1

[3x1]

4x3

[8x3]

4x8

(words)

$S_0\ S_1\ S_2\ S_3\ h_{11}\ h_{12}\ h_{13}\ h_{14}$

$S_0\ S_1\ S_2\ S_3\ h_{21}\ h_{22}\ h_{23}\ h_{24}$

$S_0\ S_1\ S_2\ S_3\ h_{31}\ h_{32}\ h_{33}\ h_{34}$

$S_0\ S_1\ S_2\ S_3\ h_{41}\ h_{42}\ h_{43}\ h_{44}$

intely of 4

1x12

[3x1]

3x3

[8x3]

1x8

(words of one)

$$C_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3 + \alpha_{14} h_4$$

Decoder $t=2$ $S_1 = [S_{11}\ S_{12}\ S_{13}\ S_{14}]$

$c_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3 + \alpha_{24} h_4$

e)

$$\begin{bmatrix} S_{11} & S_{12} & S_{13} & S_{14} & h_{11} & h_{12} & h_{13} & h_{14} \\ S_{11} & S_{12} & S_{13} & S_{14} & h_{21} & h_{22} & h_{23} & h_{24} \\ S_{11} & S_{12} & S_{13} & S_{14} & h_{31} & h_{32} & h_{33} & h_{34} \\ S_{11} & S_{12} & S_{13} & S_{14} & h_{41} & h_{42} & h_{43} & h_{44} \end{bmatrix}$$

$c_2\ S_2\ Y_1 \rightarrow LSTM \rightarrow Y_2 \rightarrow \hat{Y}/S_2$

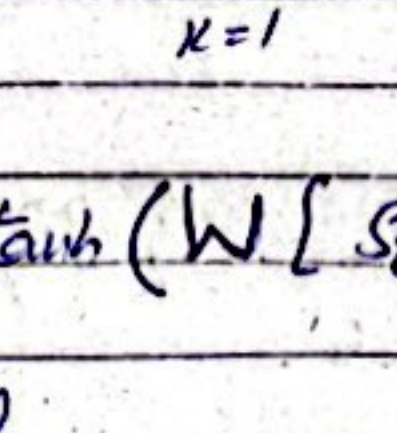
weight same ~~for~~ until all sentence passed
then next batch propagate
then next sentence send.

⇒ { decoder hidden state change only because that's why I change each time step }

time distributed ~~to~~ Feed connected neural N/W.

{ parameter update when batch propagate }

A B C D F E H



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

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \quad \text{Softmax}$$

$$e_{ij} = V_a \tanh(W[s_{i-1} \cdot h_j] + b)$$

Bahdanau Attention

also called additive attention

Neural N/w ⇒ alignment model

Luong Attention

(Multiplicative Attention)

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

Luong Attention

$\alpha_{ij} = f(s_i, h_j)$

- more dynamically output adjust
- current decoder hidden state. ⇓ updated information more than previous

not used Complex N/w

Simple $s_i \cdot h_j$ (dot product)

⇓ two vector similar

$s_i = [a\ b\ c\ d]$

$h_j = [c\ d\ e\ f]$

$$s_i^T = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \cdot [c\ d\ e\ f] = [ac\ bd\ ce\ df]$$

⇓

e_{ij}

⇓

softmax

⇓

α_{ij}

here basic funda is not how good you find approximation function, your her main funda is encoder which hidden state is useful that also do dot product.

→ dot product fast

→ recent decoder hidden state performance

Bahadanau Attention

$\alpha_{ij} = f(s_{i-1}, h_j)$

⇓

previous decoder hidden state

⇓ previous

FFNN

unnecessary parameter add/ Calculate

if two vector similar dot product more

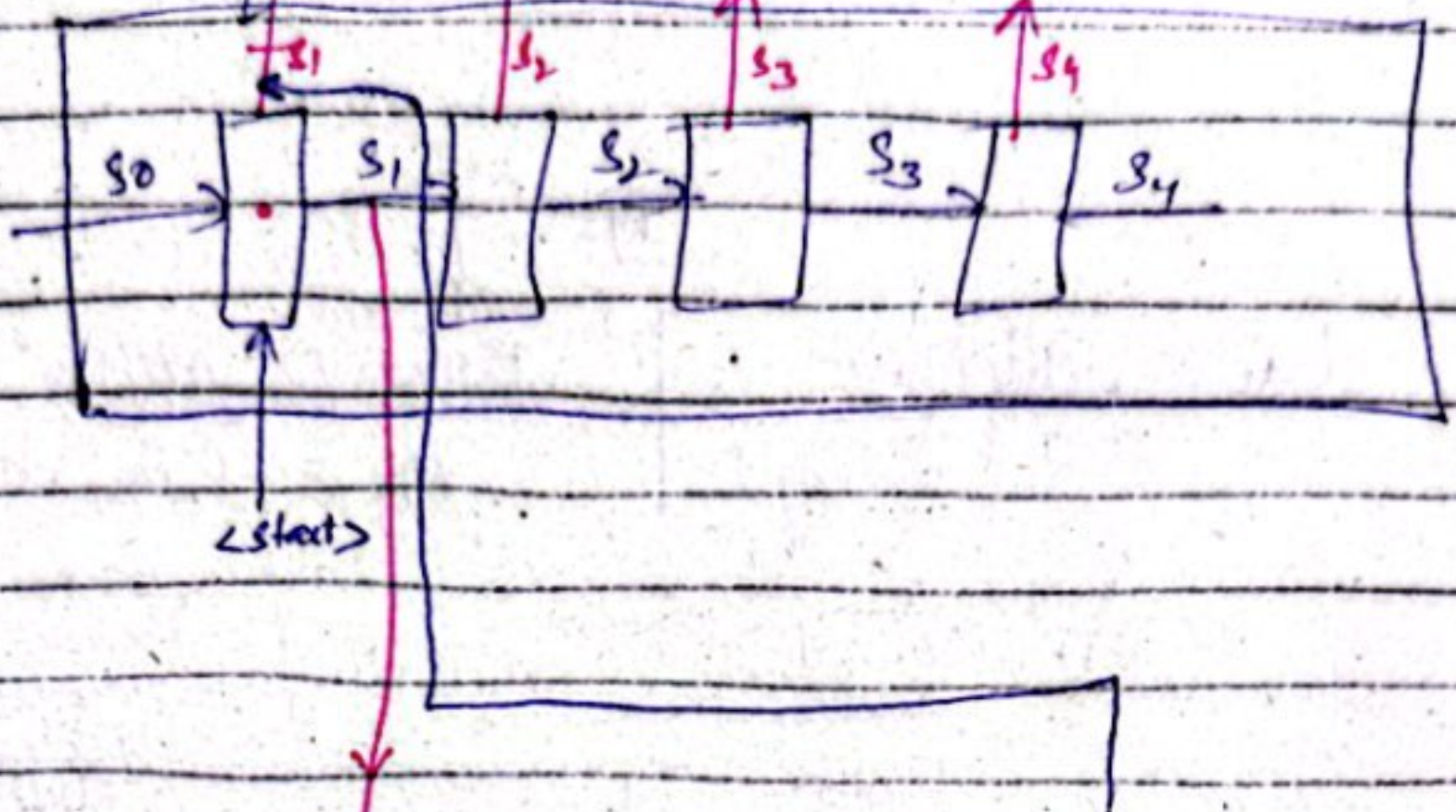
If two vector dissimilar dot product less

less no of param

updated information → recent decoder
→ dot product

$S_1 \cdot C_1 = \tilde{S}_1$ → softmax → output

Recurrent n/w



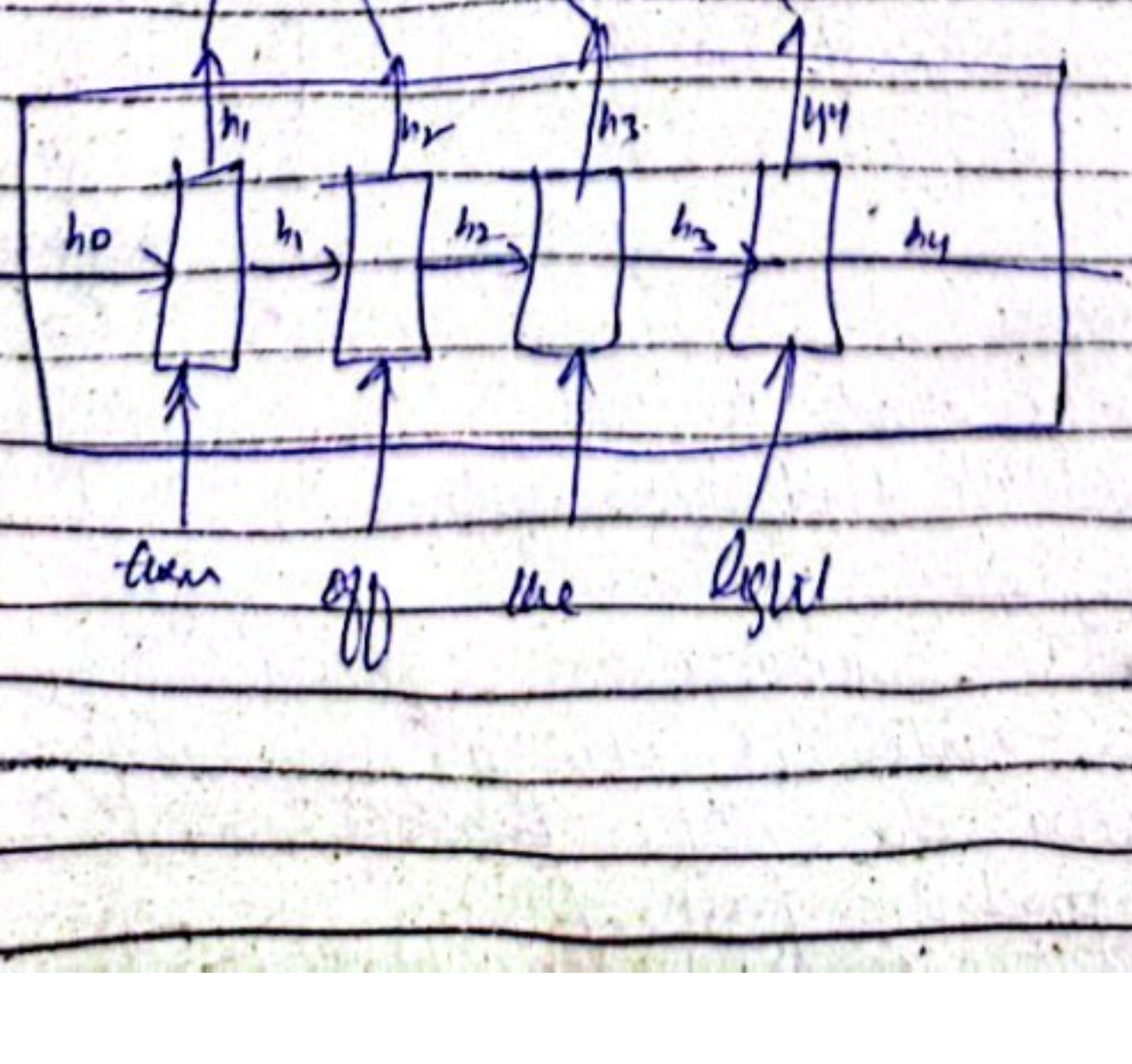
α_{11} α_{12} α_{13} α_{14} $\Rightarrow c_i = \sum \alpha_{ij} h_j$

Softmax

$[s_1 h_1 \quad s_1 h_2 \quad s_1 h_3 \quad s_1 h_4]$

$[e_{11} \quad e_{12} \quad e_{13} \quad e_{14}]$

$[\alpha_{11} \quad \alpha_{12} \quad \alpha_{13} \quad \alpha_{14}] \rightarrow c_1$



→ performance better
→ faster

Self Attention

- NLP → Application
  - → sentiment
  - → machine translation
  - → Named Entity Recognization

important requirement
how words converted into number (vectorize) efficiently

word → number (vectorization)

1- One hot Encoding

→ mat cat mat → [100] [010] [100]
→ cat sat sat → [010] [001] [001]

↳ unique word find
mat
cat
sat

↳

	mat	cat	sat
mat	1	0	0
cat	0	1	0
sat	0	0	1

this way is efficient

2) Bag Of words
→ how times a word occur
→ word occur or not
)

	unique	word		
	mat	sat	cat	
Sen1	2	0	1	7
Sen2	0	2	4	7

3) TF-IDF

4) Word Embedding
↳ word better way to represent numbers.
↳ because it has capability of semantic meaning capture.

↳ take large data
↳ neural n/w understand and which word is used in which context
and every word represent into n-dim vector

	dog	cat	person	king
king →	0.6	0.2	0.1	0.7
queen →	0.3	0.2	0.4	1.0

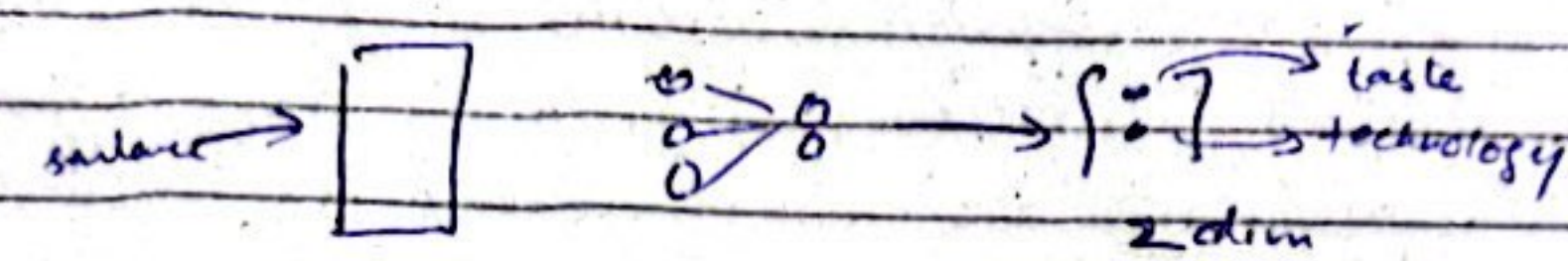
two word similar → vector similar.
each dim represent some aspect (property) about data. we can't see
each dim some meaning and each dim similarly see

somehow word meaning change into vector

The problem in embedding "average embedding"

Eg:

1) An apple a day keeps the doctor away
2) Apple is healthy
3) Apple is better than orange
4) Apple makes great phone



our word embedding does not capture meaning, it capture average meaning

Apple

	Taste	Technology
1)	[0.6	0]
2)	[0.7	0]
3)	[0.8	0]
4)	[0.8	0.2]

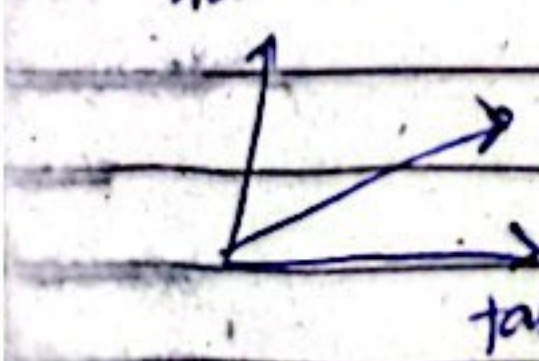
Total 10,000 word

⇒ Often sentence apple as a fruit came
⇒ 6000 " " " a company

Taste	Technology
[0.9	0.3]

↓

taste is more tilted toward taste in comparision to technology



word embedding how looks, it depend on data.

word embedding basically any particular word "average meaning" represent in vector.

word embedding make one time and used so many times.
it is static.

apple [0.9 0.3]
↓
used so many times
↓
it is problematic

e.g. application
translation.

apple → सेब

Apple launched a new fone while I was eating an orange.

first you convert each word its respective word embedding

↓

[0.9 0.3] [ ] [ ]

↳ but its accept where apple used like as fruit
but in this sentence, it is used as tech company.

ideally instead of static embedding
used contextual embedding

↓

that value dynamically
change based on content.

means if see in sentence word
"laurched" or "phone", it automatically
increase tech embedding.

[0.9 0.3]

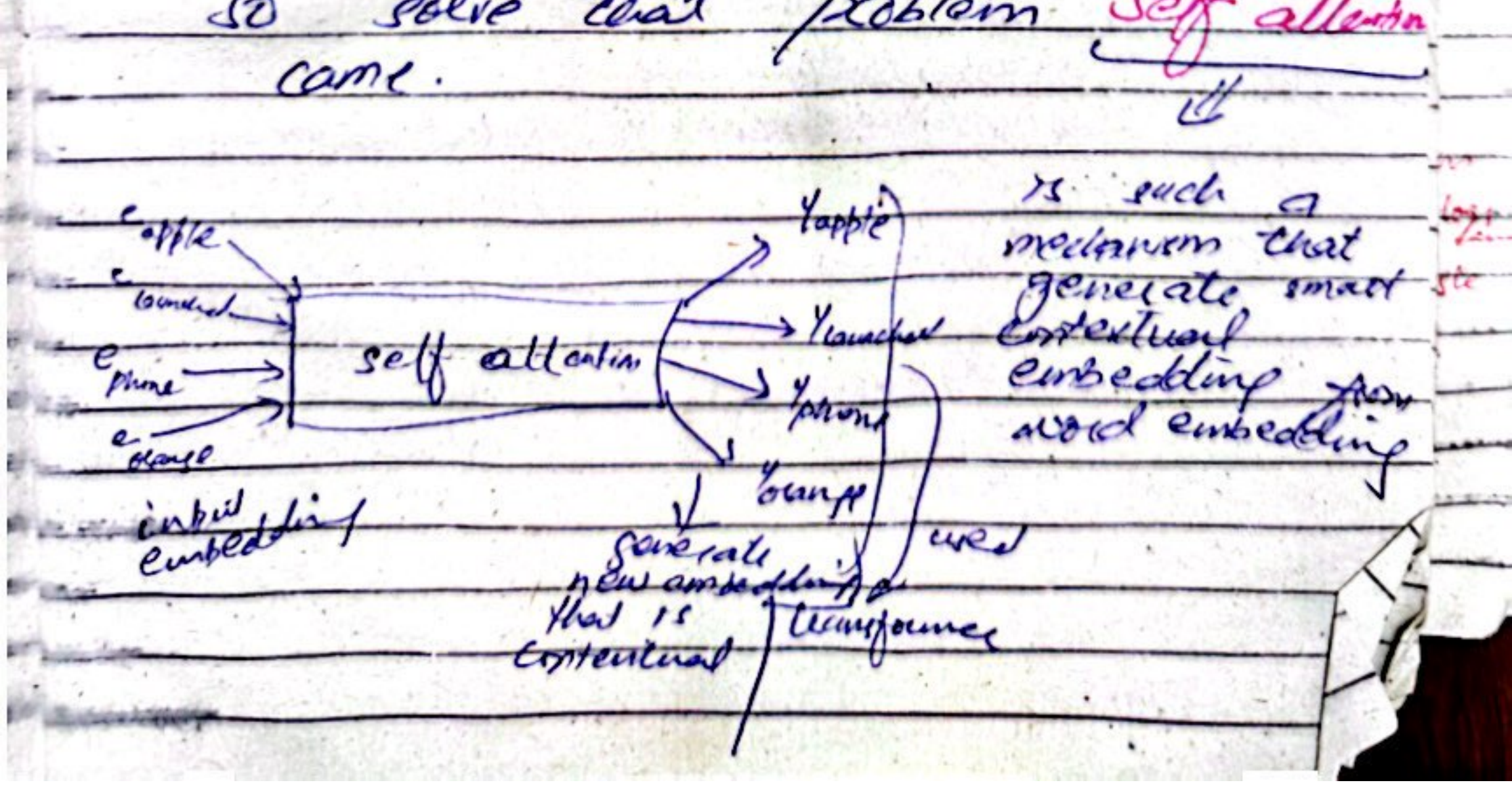
↓ ↓

[0.3 0.8]

ideally it can't confuse on "orange" word.

but our word embedding can't do
because it is static

So solve that problem Self attention
came.



Gen AI
↓
transformer
↓
self attention

SELF - Attention

Any NLP application you make
↳ How represent word into vector

Hello → [0.3 | 0.9 | 1 | 0] → one hot encoding
→ BOW
→ Embedding
↓
word semantic meaning change into word

but problem in word embedding is, it is static. once it create, you use it.

financial institution
bank
river bank

eg

Money Bank [0.6 0.2 0.1 0.7]

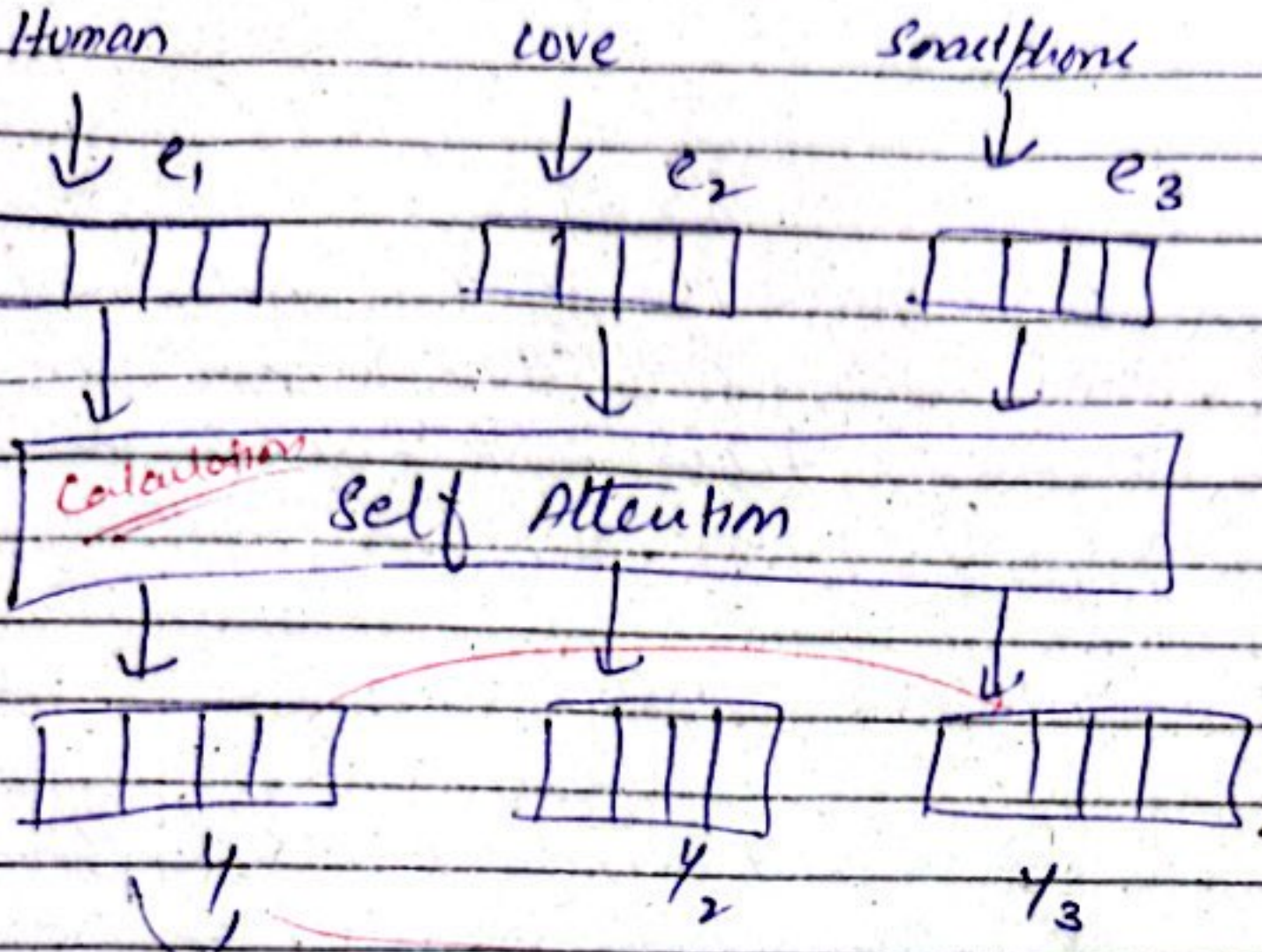
River Bank [0.6 0.2 0.1 0.7]

But bank embedding is exactly same. it is problem.

So solve this problem, we need contextual embedding.
↓
→ dynamically change
→ context-aware to content it change.

↳ self attention solve this problem

static embedding → dynamic contextual Embedding



It understand current word which context it used.

Post principle Approach

Sen 1 :- money bank grows
Sen 2 :- River bank flows

bank is same word but used in different context it's meaning is not same. but in embedding it is same

Sen 1	Sen 2
bank ≠ bank	
bank = 0.3 money + 0.7 bank + 0.1 grow	bank = 0.5 river + 0.4 bank + 0.1 flows
Now bank meaning is different.	different context

Sen 1

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.

Sen 2

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.

It's benefit is, if you focus on bank on both sentence.
L.H.S is same but R.H.S is different in both sentence.

⇒ Now do little bit change words are change into embedding.

$e^{new}_{money} = 0.7\, e_{money} + 0.2\, e_{bank} + 0.1\, e_{grows}$
$e^{new}_{bank} = 0.25\, e_{money} + 0.7\, e_{bank} + 0.05\, e_{grows}$

↓ vector → n-dim

vector → n-dim

Now what represent number 0.7, 0.2, 0.1

$e^{new}_{money} = 0.7\, e_{money} + 0.2\, e_{bank} + 0.1\, e_{grows}$

money new embedding is made up of

→ (0.7) embedding of old money +
→ 0.2 " " " bank +
⇒ 0.1 " " " grows

↓

actually it represent similarity b/w words embedding

0.7 → e_{money} e_{money}
0.2 → e_{money} e_{bank}
0.1 → e_{money} e_{grows}

embedding is vector in high dimension space

dot product b/w word embedding

(49 5) (49 2) (6 9 1)

dot product for similarity check.

24 + 2 = 25 more similar.

6 + 5 = 11

$e^{new}_{bank} = [e_{bank} \cdot e_{money}]^{0.25} \cdot e_{money} +$
$[e_{bank} \cdot e_{bank}]^{0.7} \cdot e_{bank} +$
$[e_{bank} \cdot e_{grows}]^{0.05} \cdot e_{grows}$

after dot product around.

bank → 2nd word
money → 1st word

sum = 2

$s_{21} = 36$ $s_{22} = -15$ $s_{23} = -2$

e_{money} e_{bank} e_{bank} e_{groos}

s_{21} s_{22} s_{23}

softmax

after softmax applied

w_{21} w_{22} w_{23}

X X X

e_{money} + e_{bank} + e_{groos}

$\Rightarrow w_{21} + w_{22} + w_{23} = 1$

$w_{21} = \frac{e^{s_{21}}}{e^{s_{21}} + e^{s_{22}} + e^{s_{23}}}$

$w_{22} = \frac{e^{s_{22}}}{e^{s_{21}} + e^{s_{22}} + e^{s_{23}}}$

$e^{new}_{bank} = y_{bank}$

new contextual word embedding

it → weighted sum of three embeddings

money ↓ bank ↓ grows ↓

e_{money} e_{money} e_{money} e_{bank} e_{bank} e_{bank} e_{grows} e_{grows} e_{grows}

s_{11} s_{12} s_{13} s_{21} s_{22} s_{23} s_{31} s_{32} s_{33}

e_{money} e_{bank} e_{grows} e_{money} e_{bank} e_{grows} e_{money} e_{bank} e_{grows}

s_{11} s_{12} s_{13} s_{21} s_{22} s_{23} s_{31} s_{32} s_{33}

dot product with first word

Softmax Softmax Softmax

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

$w_{11} \times e_{money} + w_{12} \times e_{bank} + w_{13} \times e_{grows}$ $= y_{money}$

$w_{21} \times e_{money} + w_{22} \times e_{bank} + w_{23} \times e_{grows}$ $= y_{bank}$

$w_{31} \times e_{money} + w_{32} \times e_{bank} + w_{33} \times e_{grows}$ $= y_{grows}$

Now we calculate new contextual embedding of each word in sentence.

Point to consider :-

→ This is parallel operation
↳ no one depend on each other.
3rd not depend on 2 or 1 word.

→ There are no learning parameter involved.
↳ no learning based on task specific.
↳ it is general.

your sentence have 20 or 200 words, you at the same time contextual embedding generate as many as you want.
↓
entire thing is parallel
↓
Speed high

Disadvantage

→ sequence info loss

first bank come at money ..

hello → e_1 → y_1

↳ it can't learn anything from data.

it learn from current sentence.

it is general contextual embedding not task contextual embedding)

↓

piece of cake ⟺ بہت آسان ہے

↓ ↓ ↓

e_1 e_2 e_3

↓ ↓ ↓

y_1 y_2 y_3

but in any data it can't generate task contextual embedding

it is general, it work on many cases. but it can't work where task contextual.

break a leg ٹانگ توڑ دو

باقاعدہ سے خوش آمدید

Only contextual embedding generation is sufficient.
We want task contextual embedding.

sentiment problem ⇒ based on sentiment problem it generate contextual embedding

translation problem ⇒ based on translation problem, it generate contextual embedding

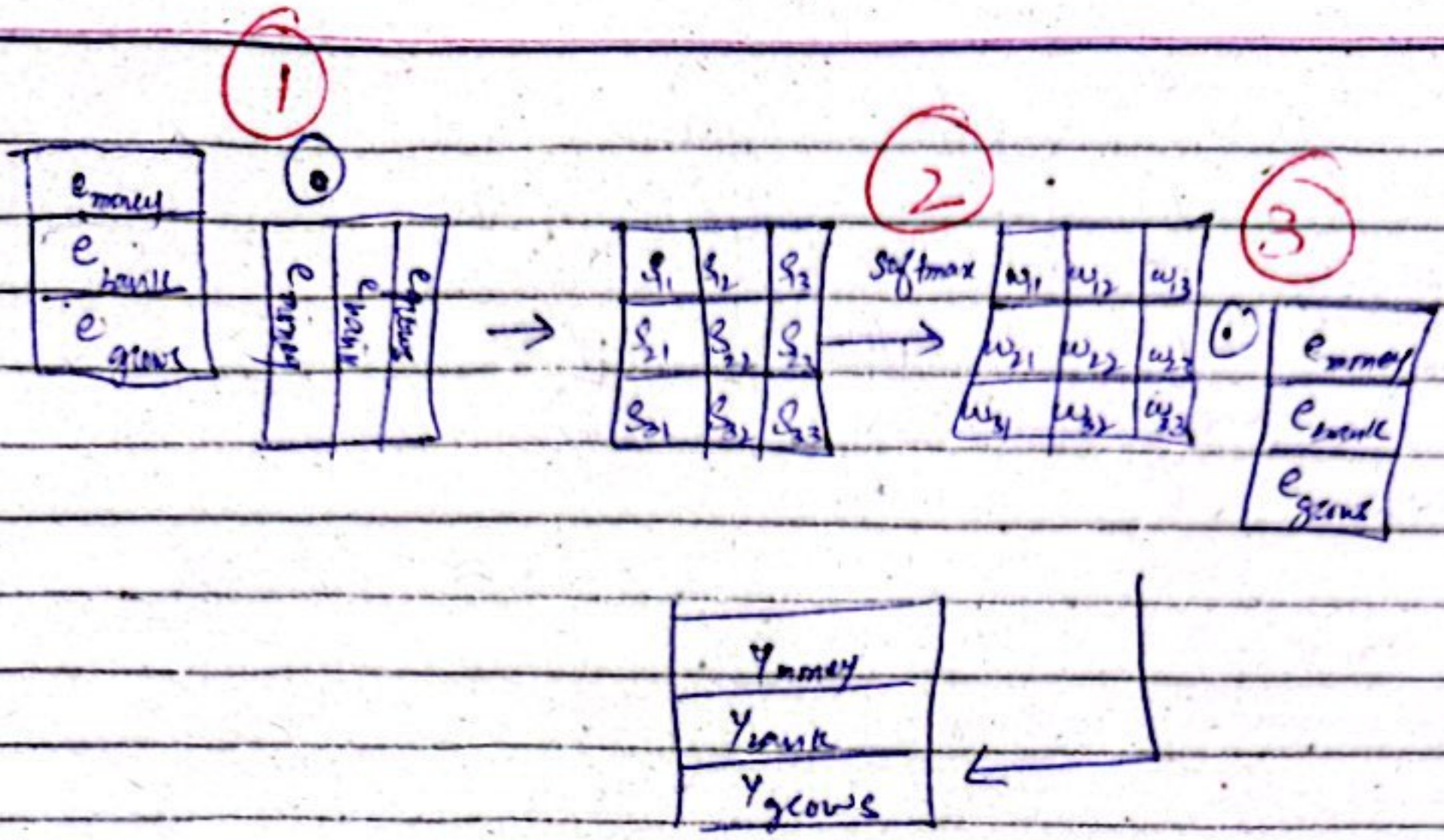
we add some weight & biasis, self attention problem learn, in this data that words how used.
that learning based on data not only one sentence.

Progress (Reverse)

word → number → embedding (semantic
- not capture contextual
  e.g river bank
  money bank

⇓

Contextual embedding
→ self attention
- general — NO parameter
- task — parameter



where learnable parameter add?

1) dot product ?
2) Softmax ✗ no weight involve
3) dot product ?.

only 1 and 3 where weight are involve

When we see e_{money} only

e_{money} e_{money} + e_{money}

①

↓

it ask remaining all embedding

how much similarity b/w you and me.

Query

↓

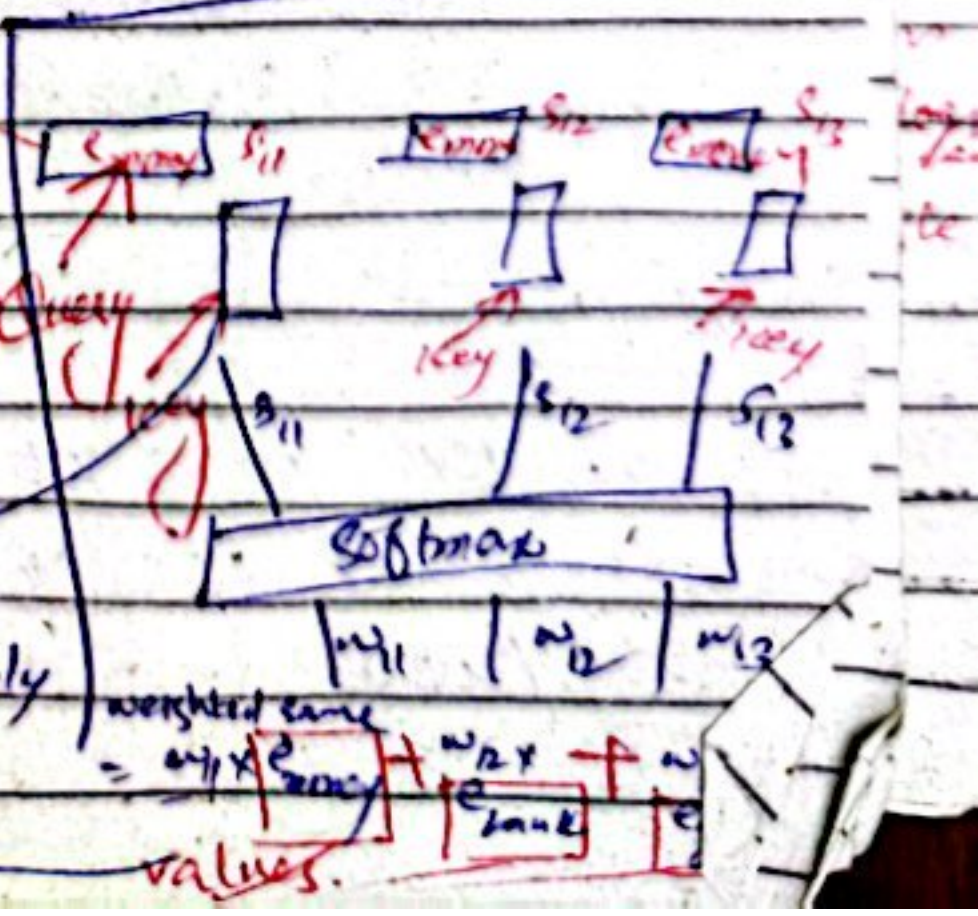
when query ask question, reply

answer, that is "key"

that. it answer is "value"

When we e_{bank} only

e_{bank} e_{bank} + e_{bank}



Dict

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

value value value

↓ ↓ ↓

key key key

d[a] = ? 2

query

So embedding we used in three ways key, query, value

money

e_{money}

e_{money}

e_{bank}

key query value key query value

During

Contextual embedding calculation process, every embedding vector three roles play

1- key

2- Query

3- value

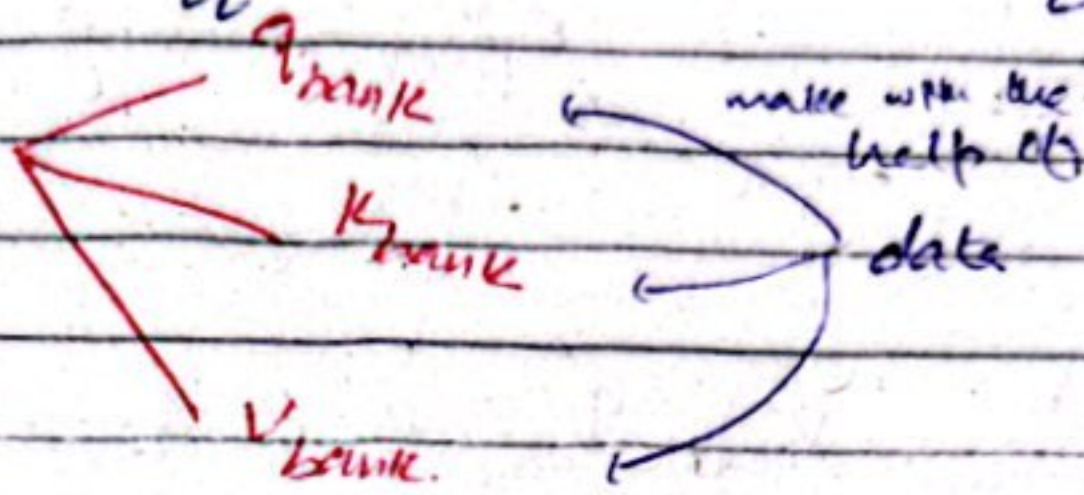
Problem

key :- that vectors convert make e_{money}

value :- " " " " e_{money}

Query :- " " " " e_{money}

ideally three different vector made from it



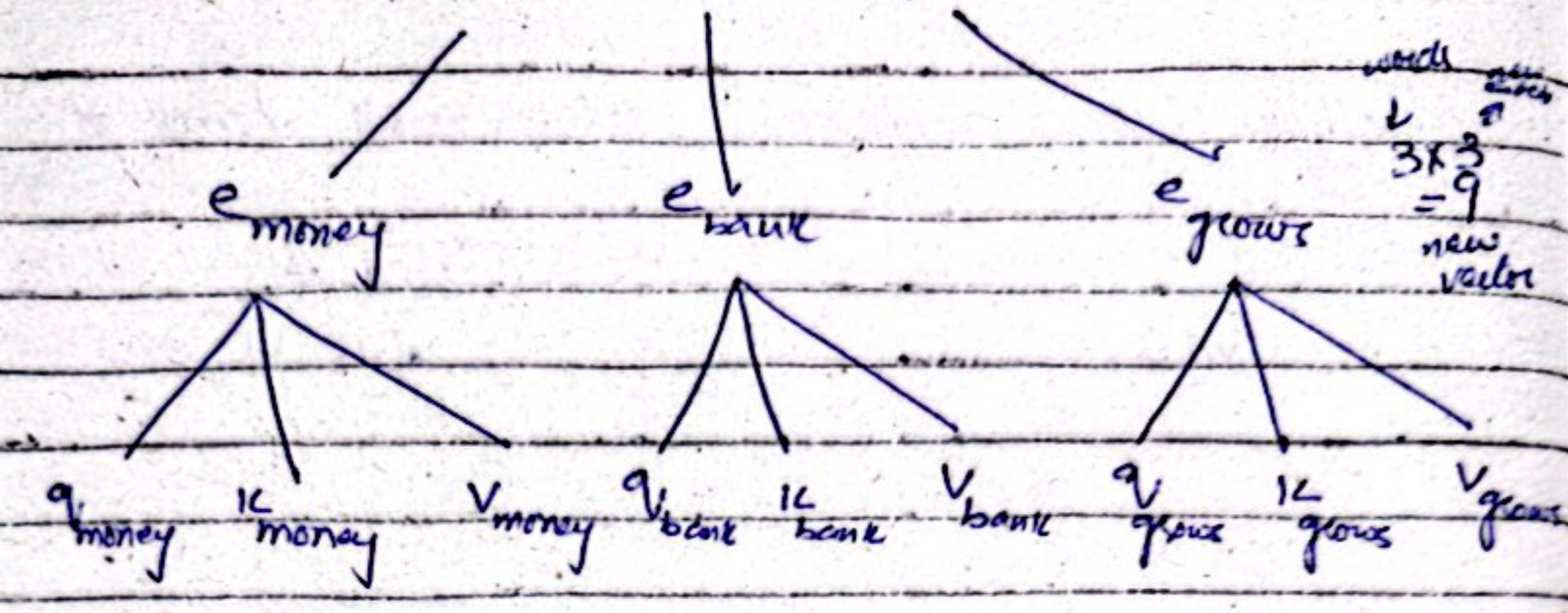
Q what benefit from it?

Ans Seperation of concerns.

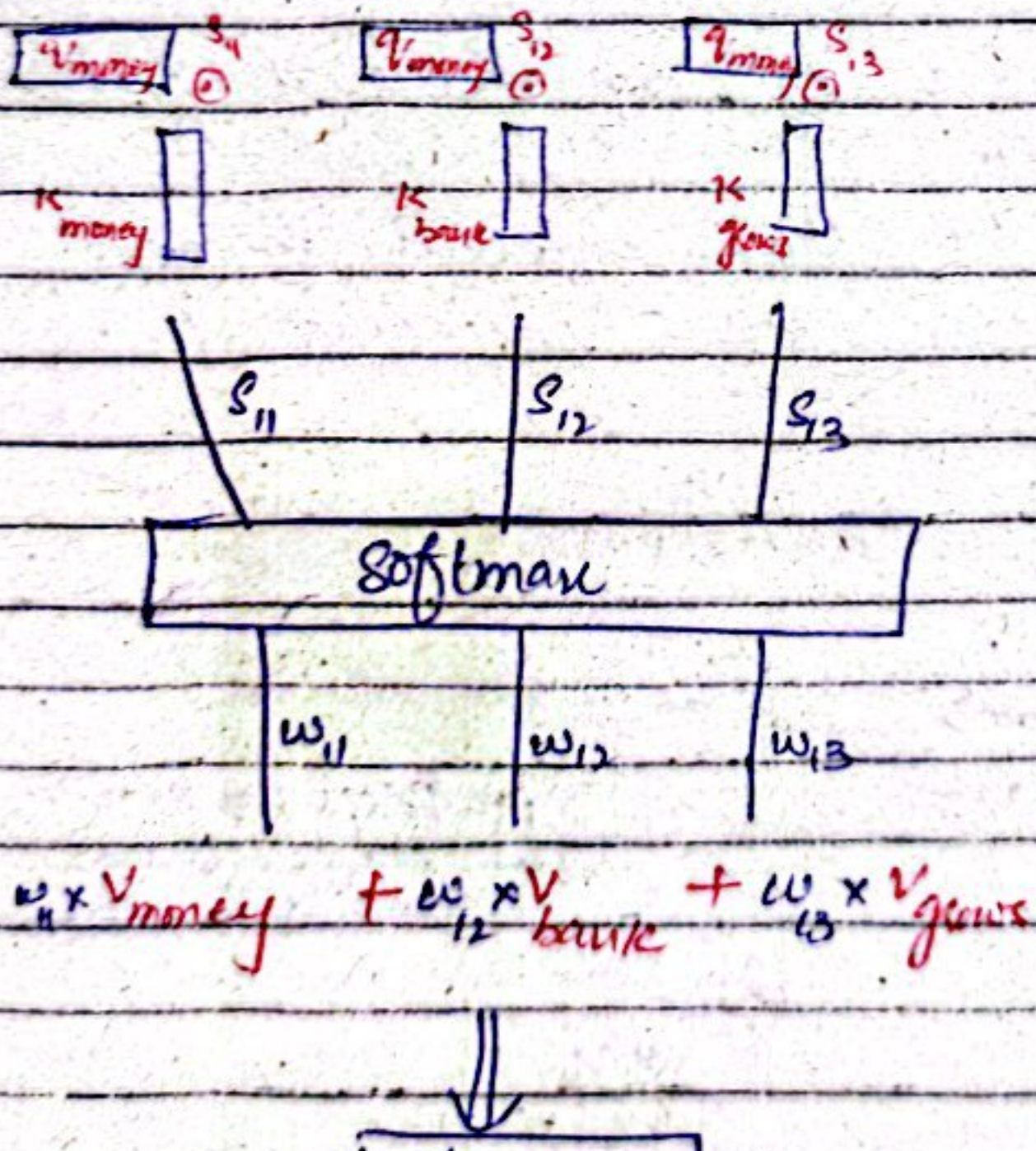
if I put all work on one vector, it can't perform perfectly.
but I transform it three different vector, it work ~~or~~ own work perfectly.

example

money bank grows



money contextual embedding

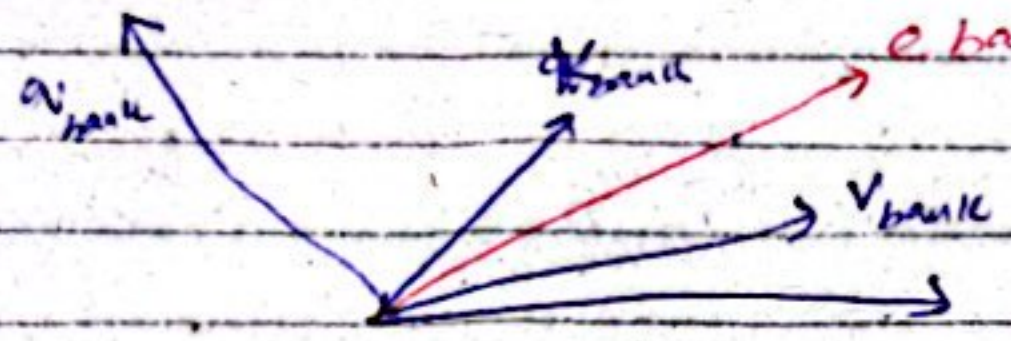


$w_{11} \times V_{money} + w_{12} \times V_{bank} + w_{13} \times V_{grows}$

Y_{money}

~~How~~ grew embedding, new query, key and value vector make.

⇒ we know we take help of data.



→ To make new vectors from one vector

(1) Magnitude (increase or decrease vector) Scaling

(2) Linear transformation

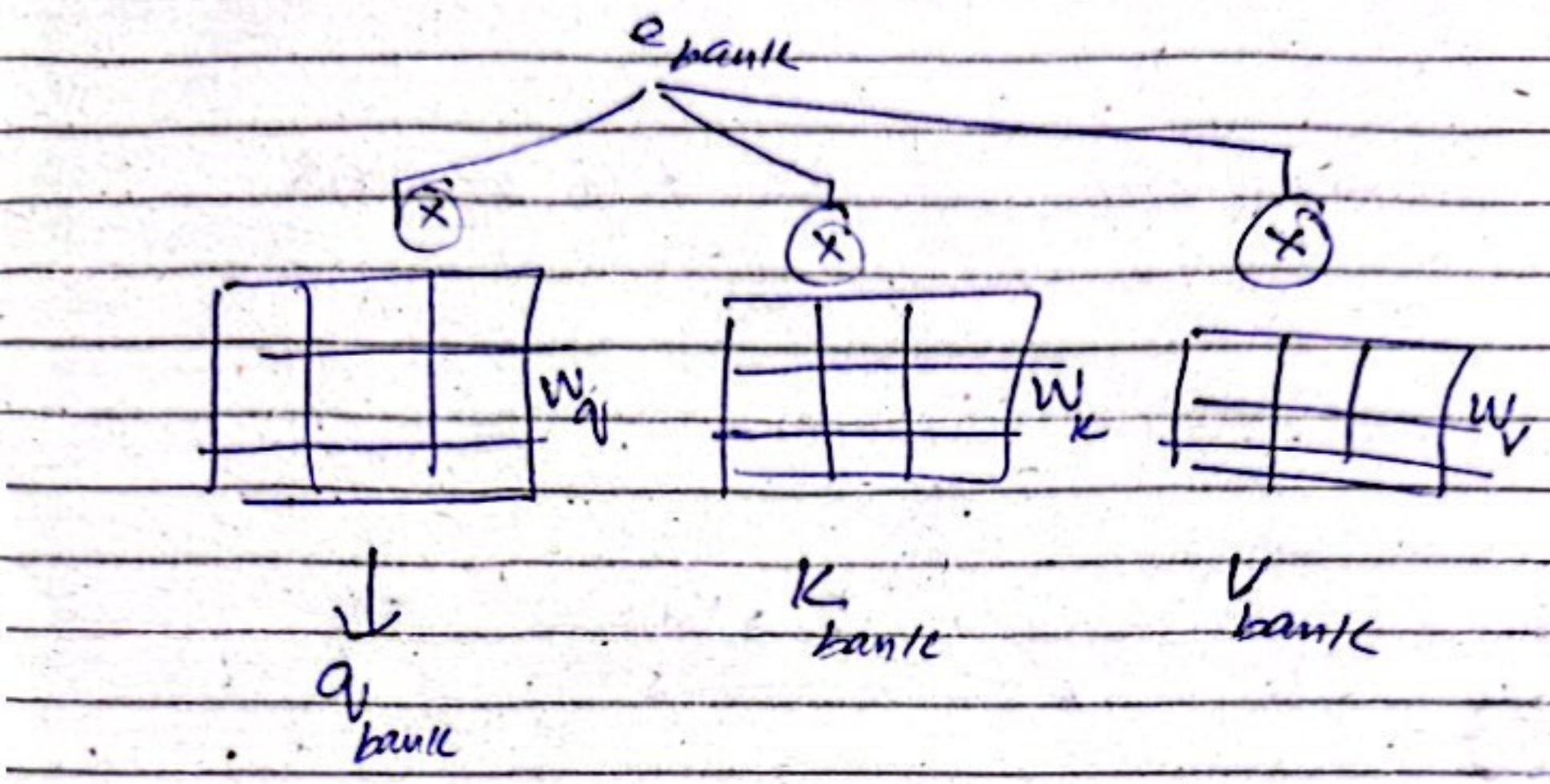
vector x [ ] matrix

↓

magnitude and direction both change

Scaling is not best, so linear transformation is best in our case.

$e_{bank} \times$ [ ]



What values are in matrix?

W_q, W_k and W_v

⇒ we decide new vector of made from original embedding by the help of data.

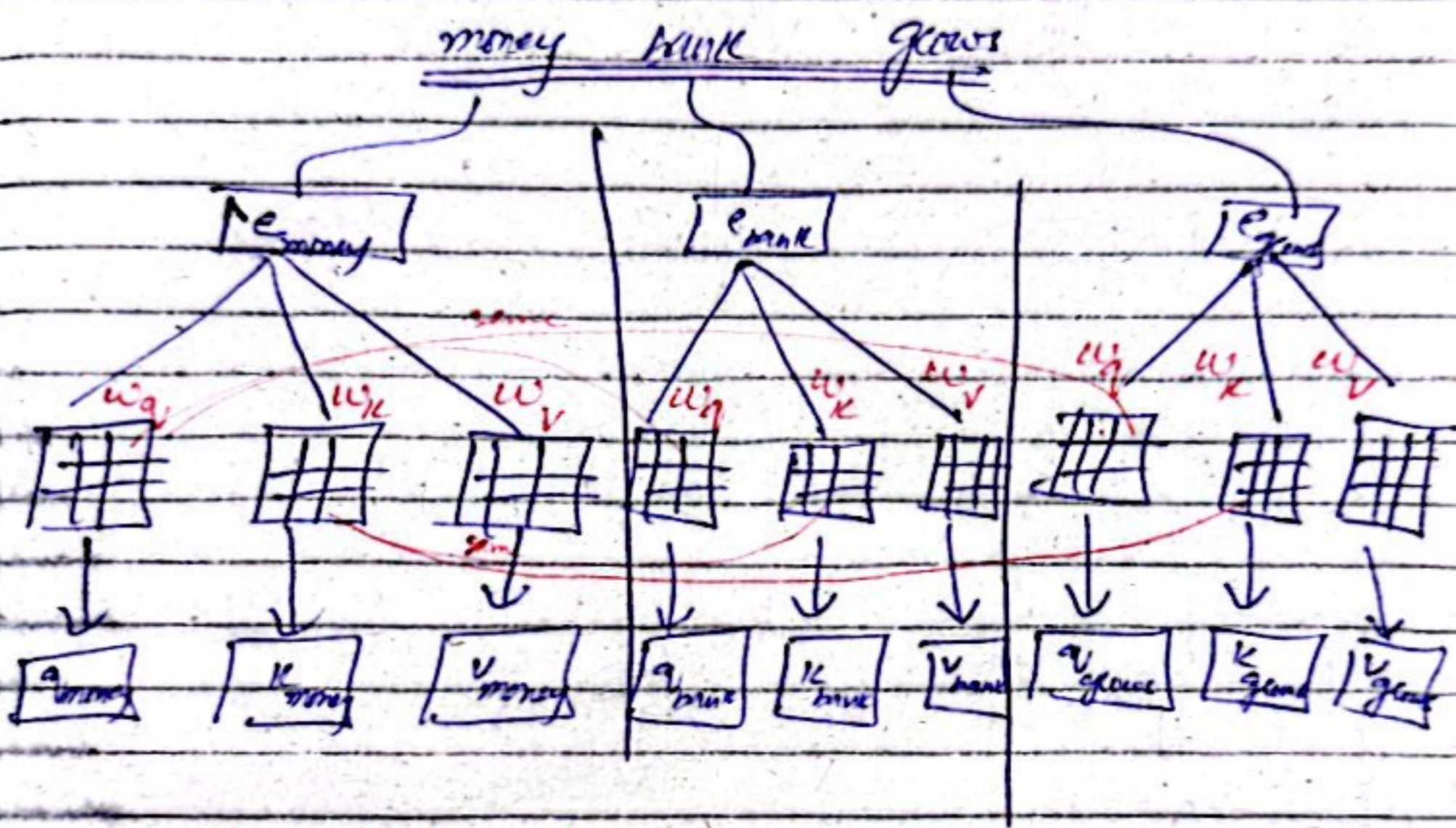
⇒ we not decide what number is matrix, training process decide

what we will do?

We will start with random value (weights)

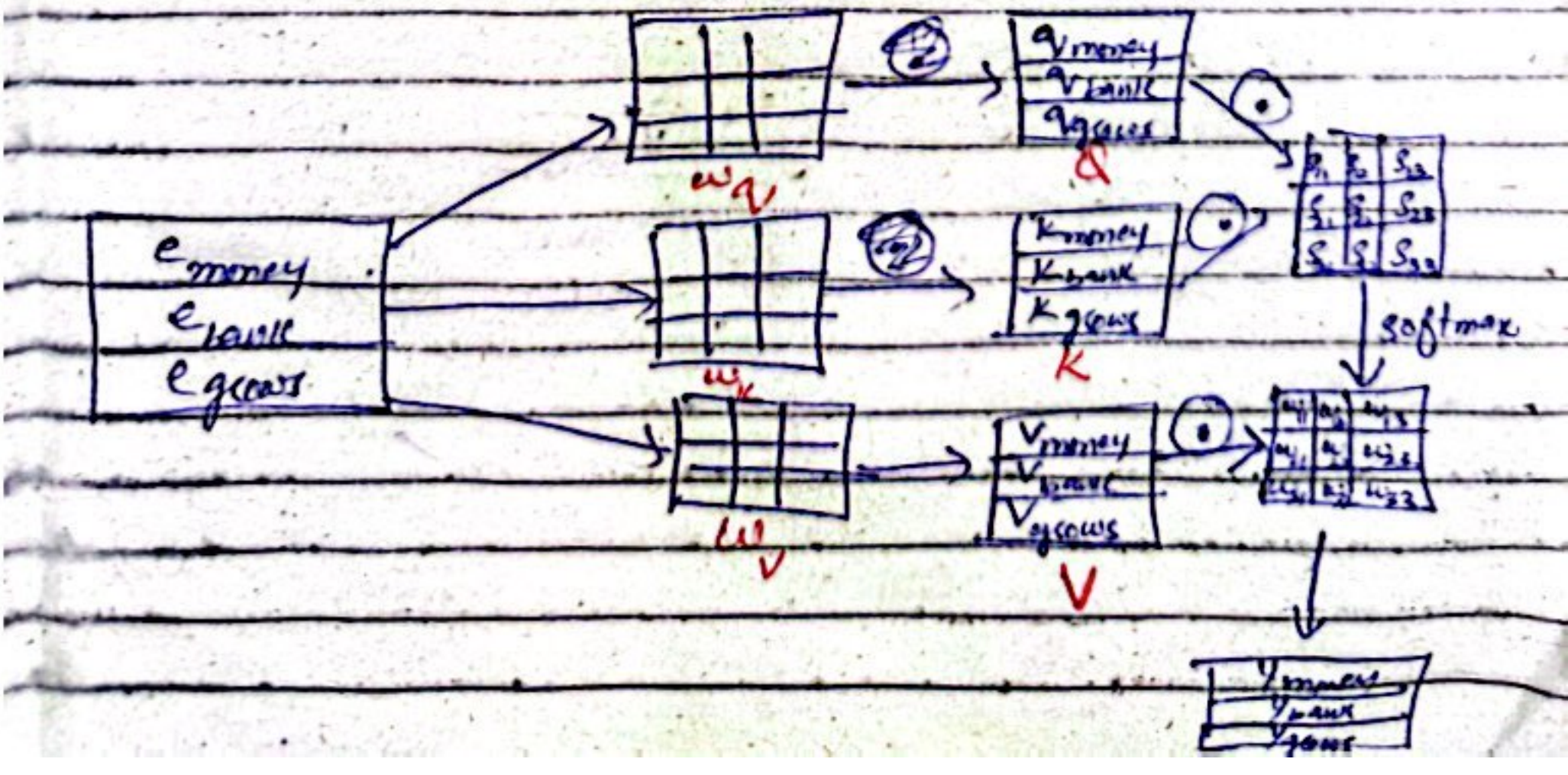
and it generate query, key, and value that runs "translation" occur some mistake occur, back propagate weight update and next sentence goes and so on..

and after learning it gain context value!



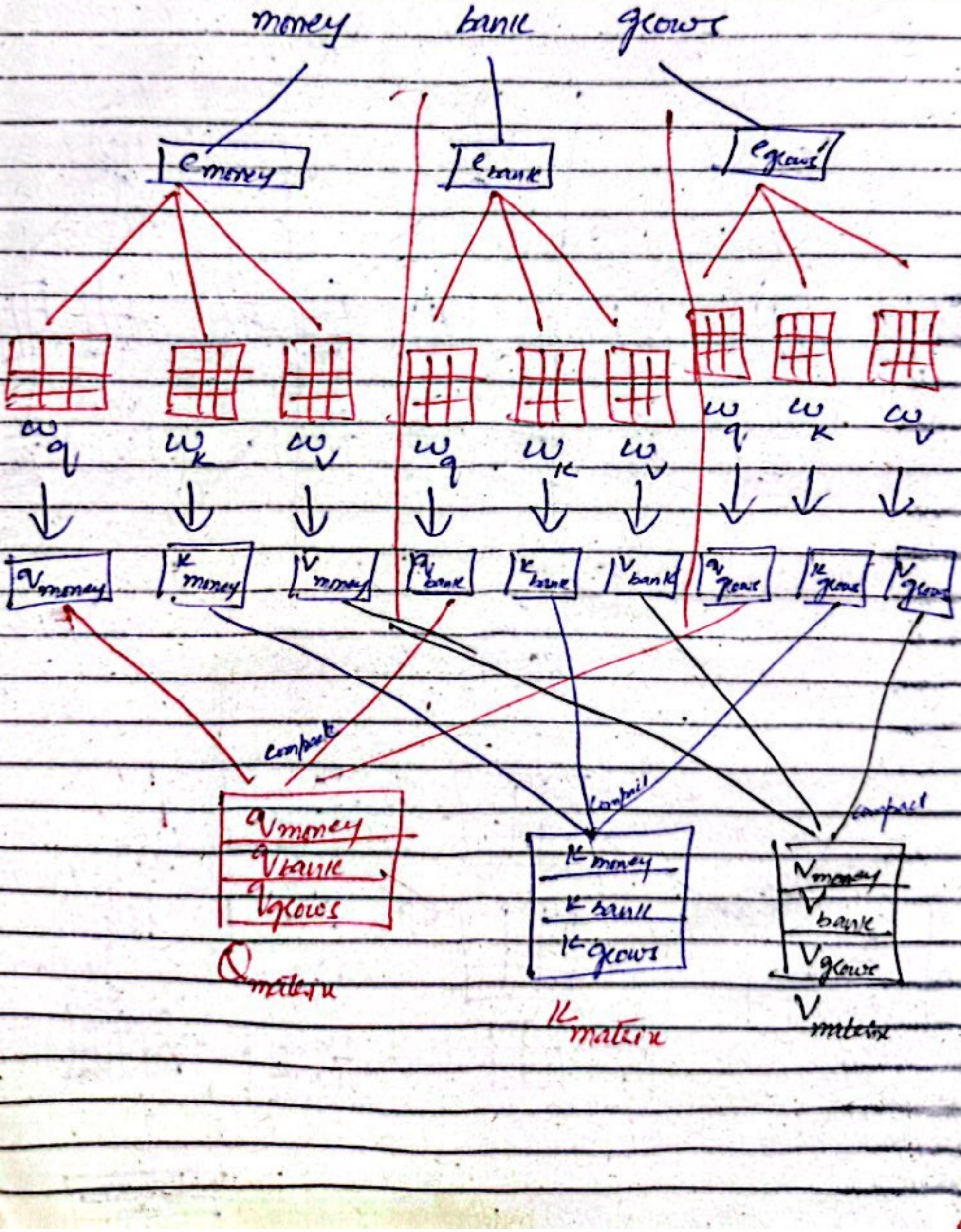
all words have same w_q, w_k, w_v. These are parameter learnable during training

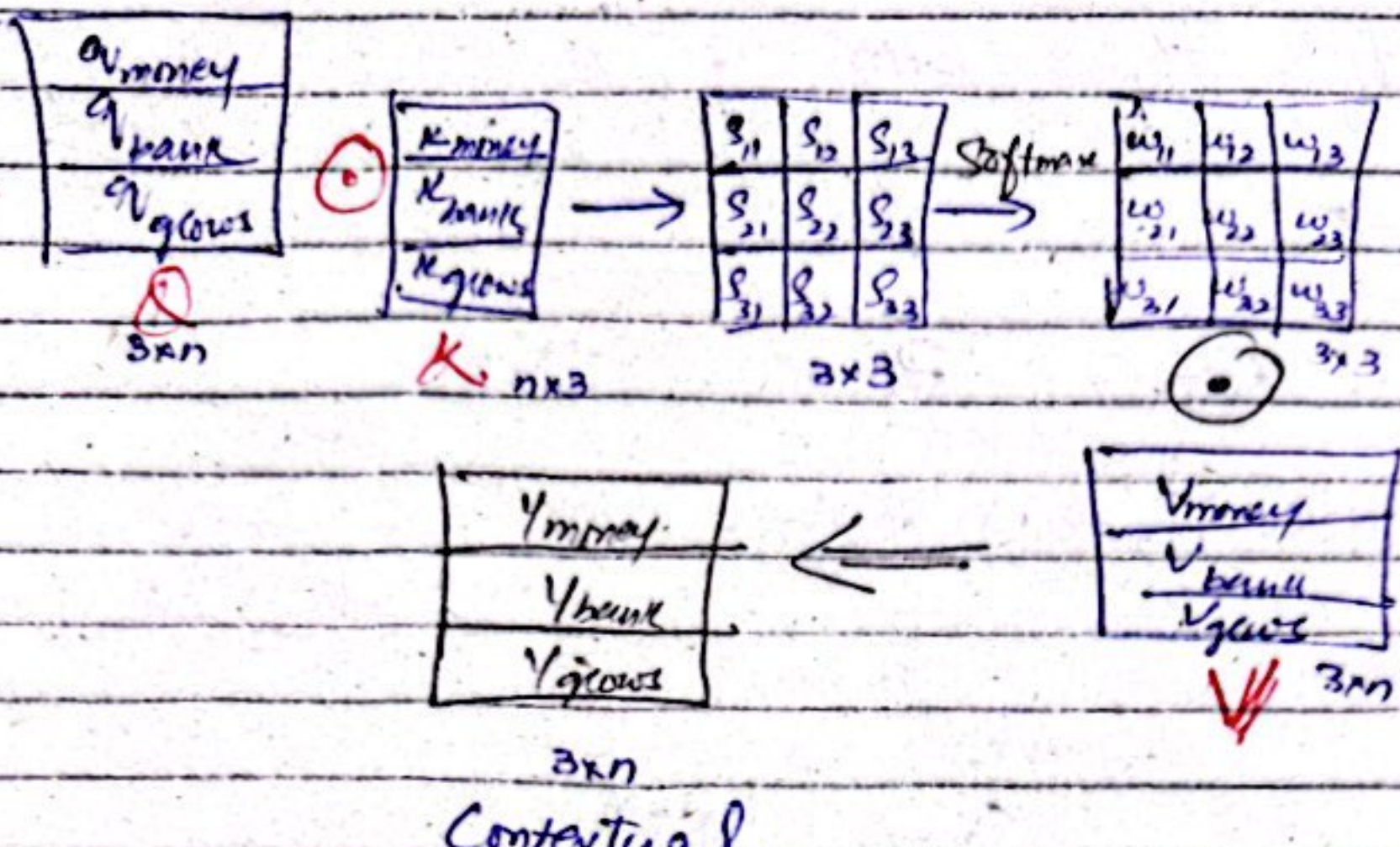
it also parallel create



Scaled Dot product Attention

Recap





Contextual embedding

$$\underset{(Q,K,V)}{\text{attention}} = \left[\{\text{Softmax}(Q \cdot K^T)\} \cdot V\right] \quad \ldots$$

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

Scaled Dot product production

$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

Unstable gradient avoid/Learning unstable

Scale

why scale and why for d_k?

→ positional encoding
→ transformer architecture

K $d_K = 3$

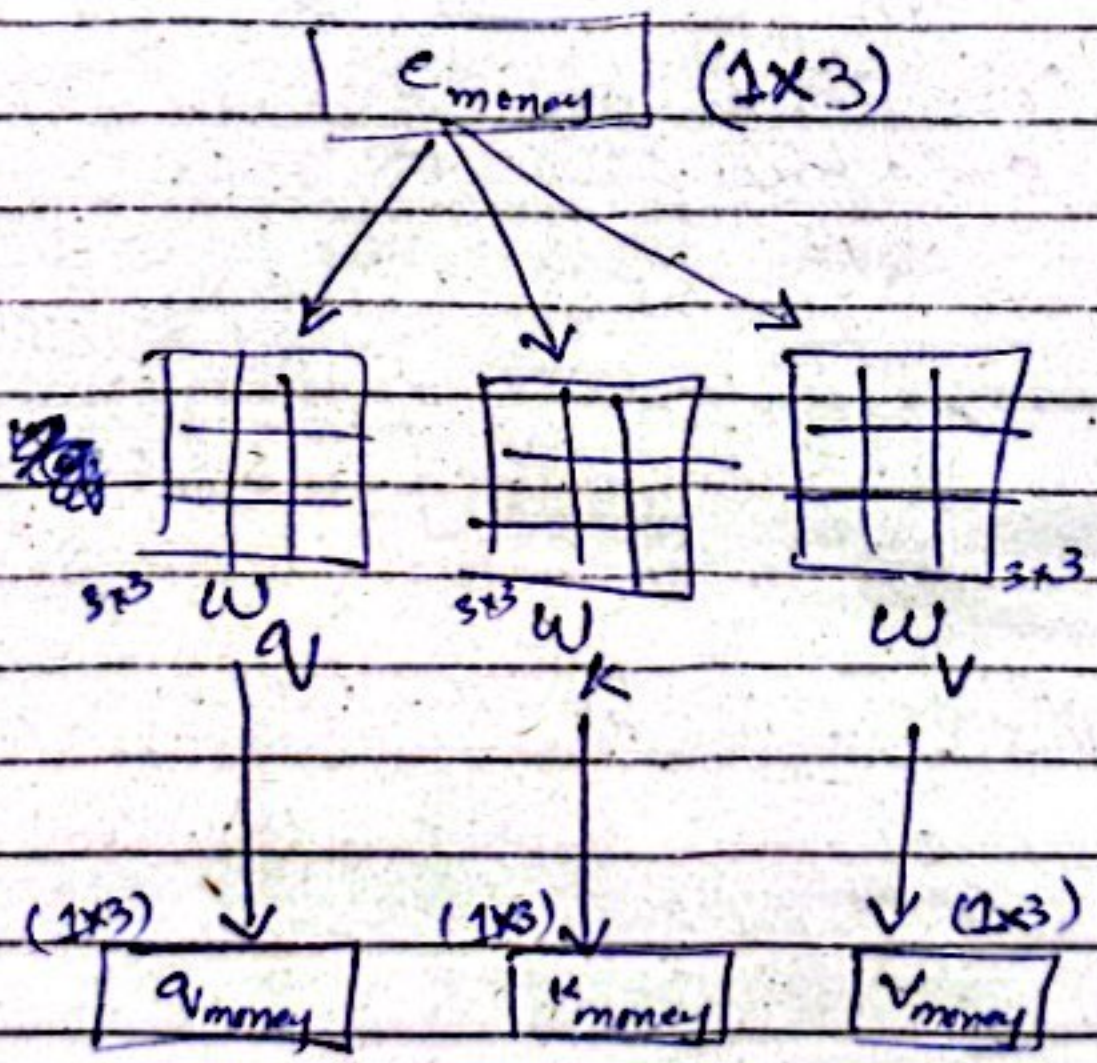
d_K = dimensionality of K-vector

e

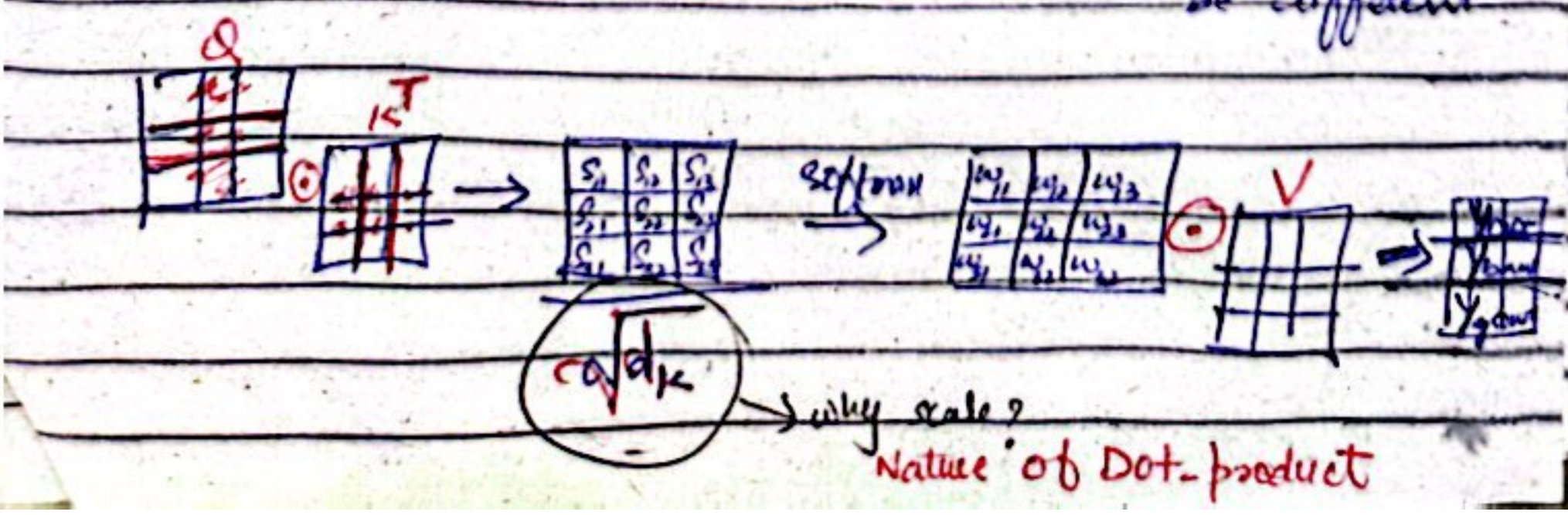
emoney B dim (1x3)

$d(w_q) = d(w_k) = d(w_v)$ it also different

$d(w_q) \neq d(w_k) \neq d(w_v)$



$d_q = d_k = d_v$ ideal case and it may be different



made of

Q → 3 vectors

K → 3 "

3×3 ⇒ (9) vector dot product

↓ 9 numbers

⇓

mean (μ)

variance (σ^2)

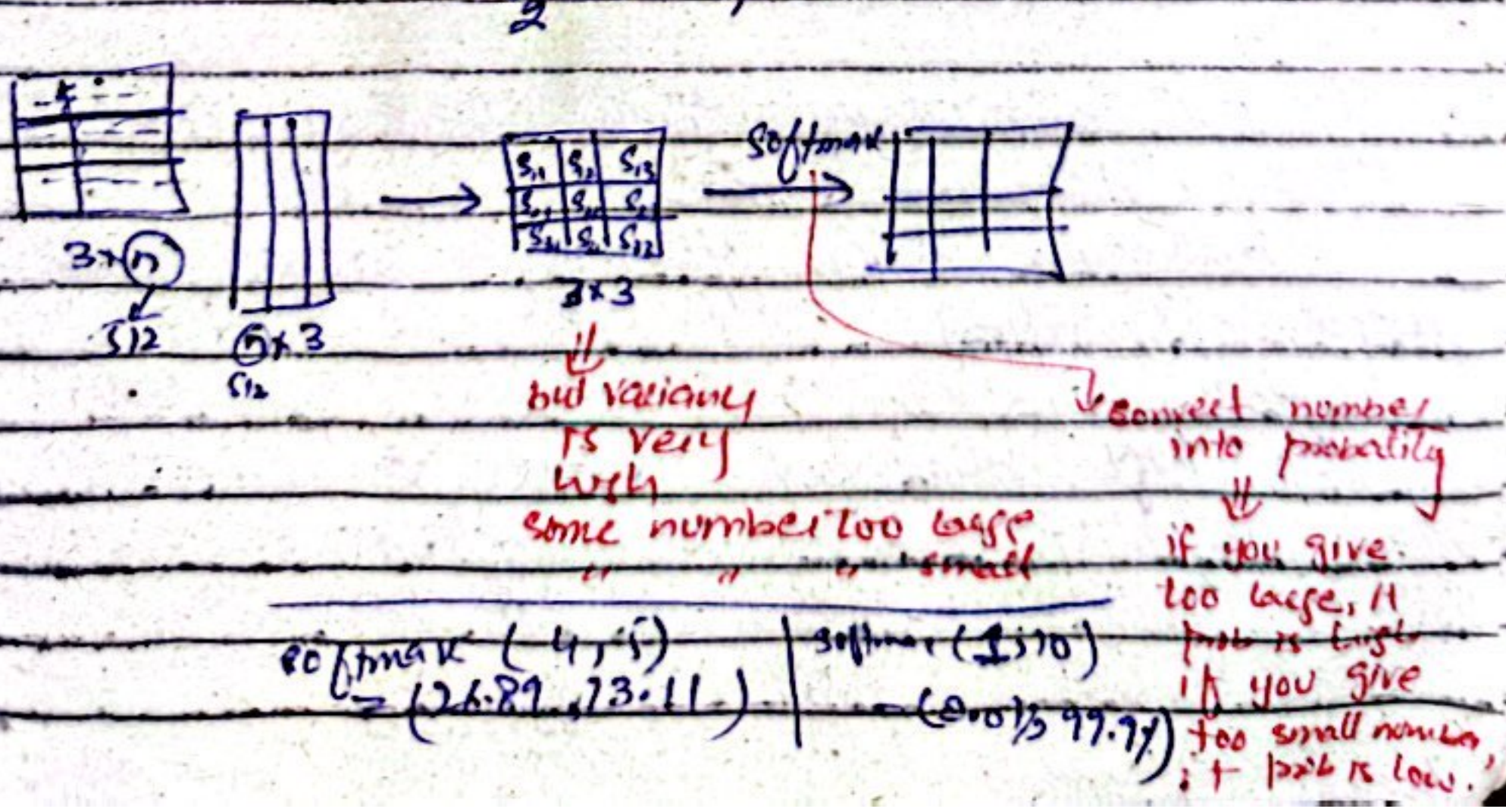
Nature of Dot product

low dim → dot product → low variance

high " → " " → high "

2-dim	3-dim
1) [1 2] ⊙ [3,2] = a	1) [- - -] . [- - -] = d
2) " ⊙ " = b	2) " " = g
3) " ⊙ " = c	3) " " = h
4) " ⊙ " = d	4) " " = i
5) " ⊙ " = e	5) " " = j
σ_1^2	σ_2^2

$\sigma_2^2 > \sigma_1^2$



If some number is too large, softmax give that number high prob and some number give too low prob

Gab too many exist, problem came, we Train whole process, it mean back propagate, gradient calculate and that bads weight update.

whatever it see large number, whole training try to correct that number and smaller number ignore, vanishing gradient problem occur, parameter never update, training process destroy.

to reduce variance, automatically when apply softmax, such prob assign near to approximately equal and training correctly occur.

To solve that problem:

↳ you can't take high dimension, take low dimension embedding
but in case you can't allow more useful information.

↳ high dimension → high variance
ll problem
low variance

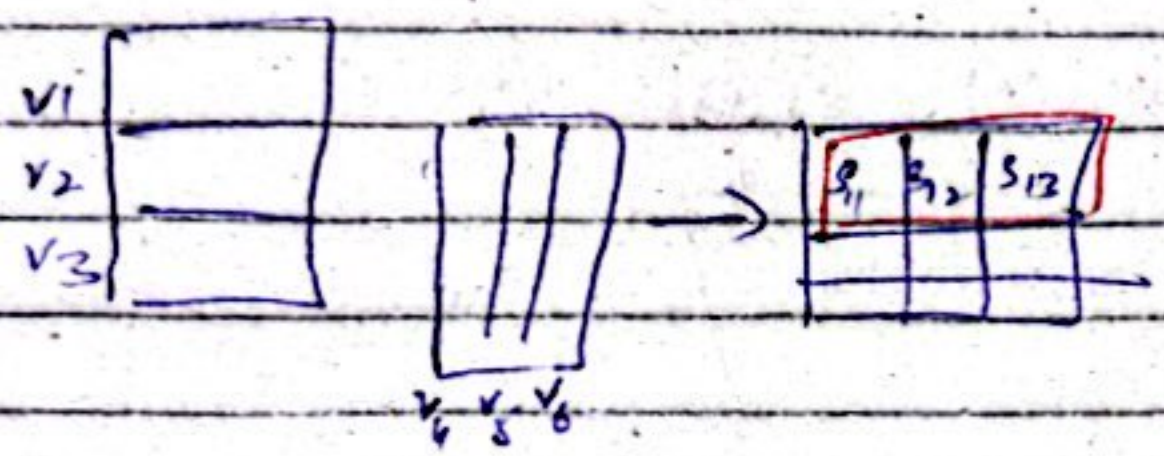
$x = 10, 20, 30, 400, 50, 60, 70$

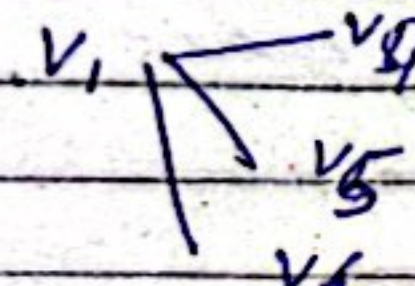
$Var(x) = 400$

$x = 1/10, 20/10, 30/10, 40/10, \frac{50}{10}, \frac{60}{10}, \frac{70}{10}$

$x = 1, 2, 3, 4, 5, 6, 7$

$Var(x) = 7$





The softmax function has smaller changes (or gradients) when the input are large. Small gradients make it harder for the model to adjust and improve during training.

why used d_k ?

1) The magnitude of dot matrix product grows with d_k.

2) larger dot products can cause (softmax activation) to push values into Regions of small gradients, making optimization harder.

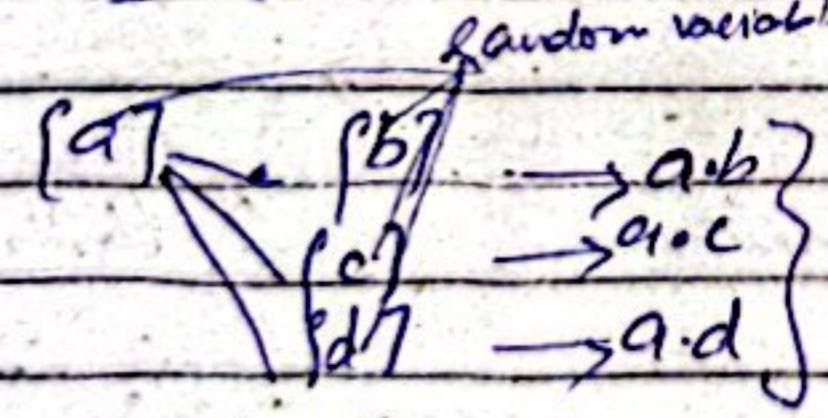
⇒ one value becomes much larger than the other.

* Softmax will treat it most important and ignore smaller values.

* Gradient becomes very small.

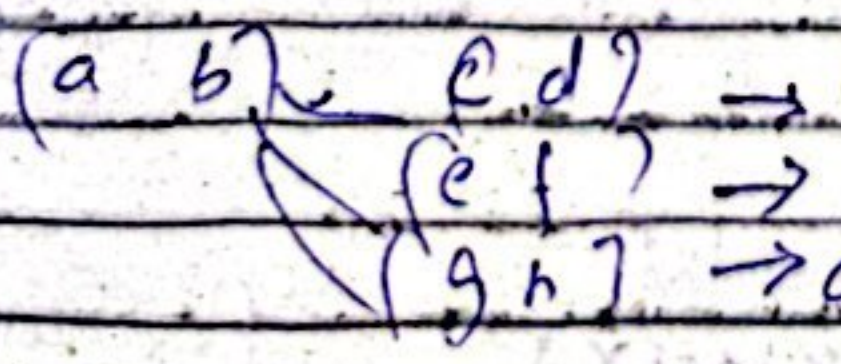
→ Var(x) ↓ small

1-dim

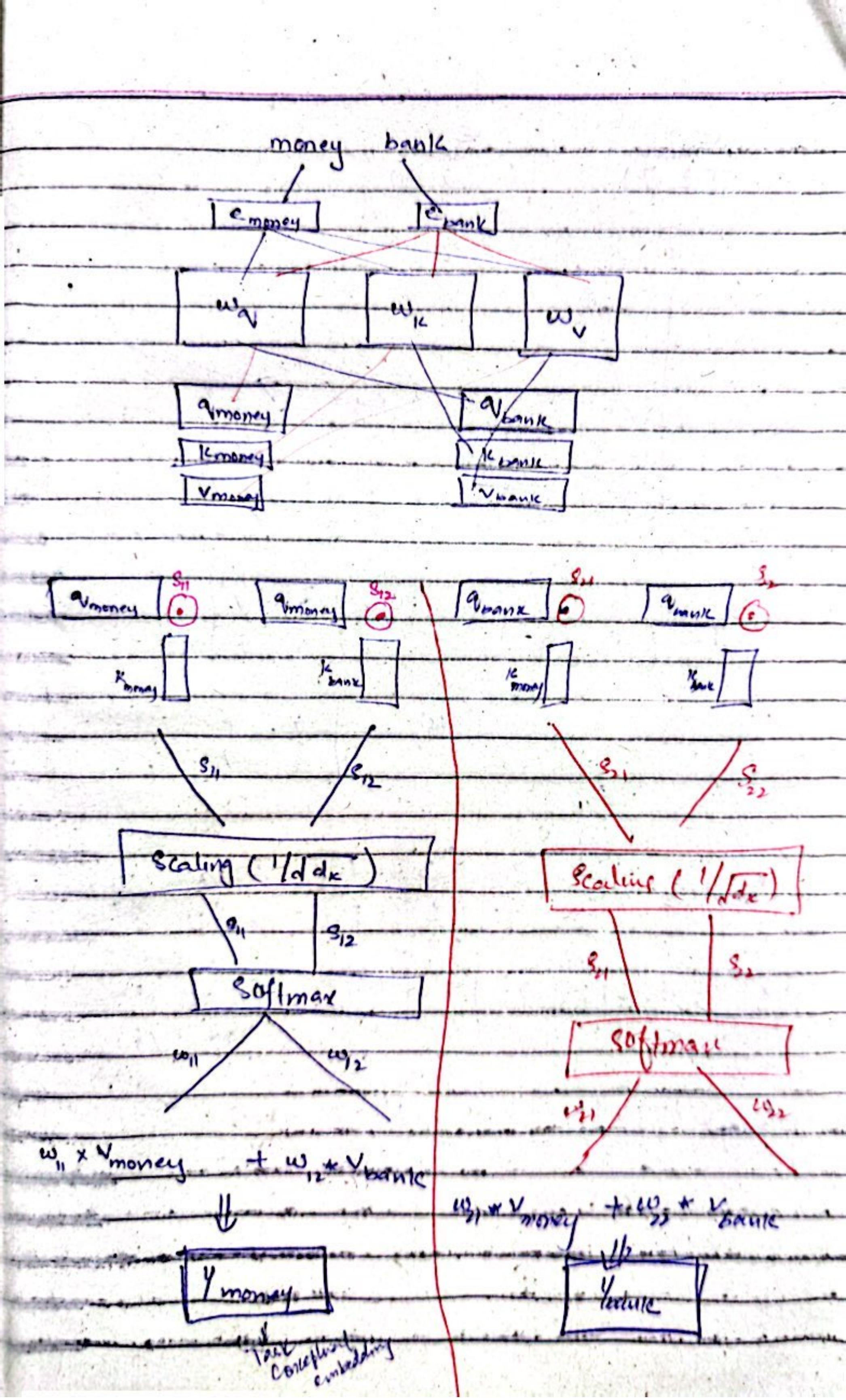


variance simply expected (val)

2-dim

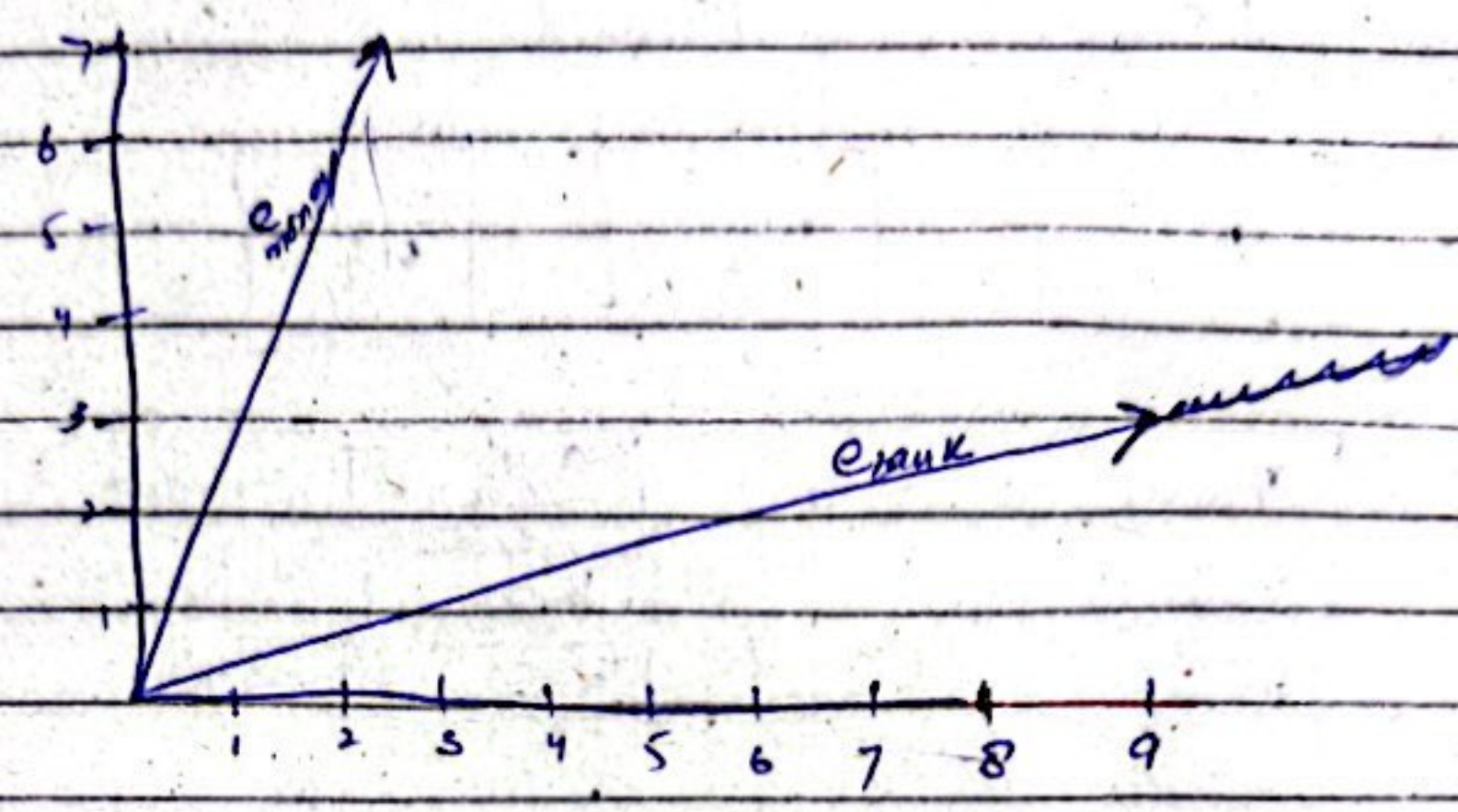


$Y \rightarrow Var(Y_{ou})$


money
bank
e_{money}
e_{bank}
w_v
w_k
w_v
q_{money}
q_{bank}
k_{money}
k_{bank}
v_{money}
v_{bank}
s_{11}
s_{12}
s_{21}
s_{22}
Scaling ($1/d\,d_k$)
Scaling ($1/\sqrt{d_k}$)
s_{11}
s_{12}
s_{21}
s_{22}
Softmax
Softmax
w_{11}
w_{12}
w_{21}
w_{22}
$w_{11} \times v_{money} + w_{12} * v_{bank}$
$w_{21} * v_{money} + w_{22} * v_{bank}$
Y_{money}
$Y_{venture}$
Text Conceptual embedding

e_{money} [2, 7)

e_{cake} (9, 3)



$$w_q = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad w_{10} = \begin{bmatrix} 3 & 4 \\ 5 & 1 \end{bmatrix}$$

$$w_v = \begin{bmatrix} 9 & 1 \\ 2 & 1 \end{bmatrix}$$

$$e_{money} \cdot w_q = [2 \quad 7] \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = [4+7 \quad 2+4] = [11 \quad 8]$$

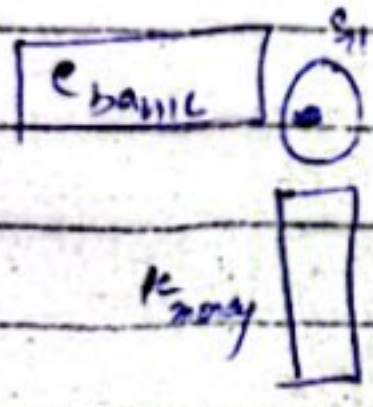
$$e_{money} \cdot w_{10} = [2 \quad 7] \begin{bmatrix} 3 & 4 \\ 5 & 1 \end{bmatrix} = [6+35 \quad 8+7] = [41 \quad 15]$$

$$e_{money} \cdot w_v = [2 \quad 7] \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix} = [8+14 \quad 2+7] = [22 \quad 9]$$

$$e_{bank} \cdot w_q = [9 \;\; 3] \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = [18+3 \;\; 3+6] = [21 \;\; 9]$$

$$e_{bank} \cdot w_k = [9 \;\; 3] \begin{bmatrix} 3 & 4 \\ 5 & 1 \end{bmatrix} = [27+15 \;\; 36+3] = [42 \;\; 39]$$

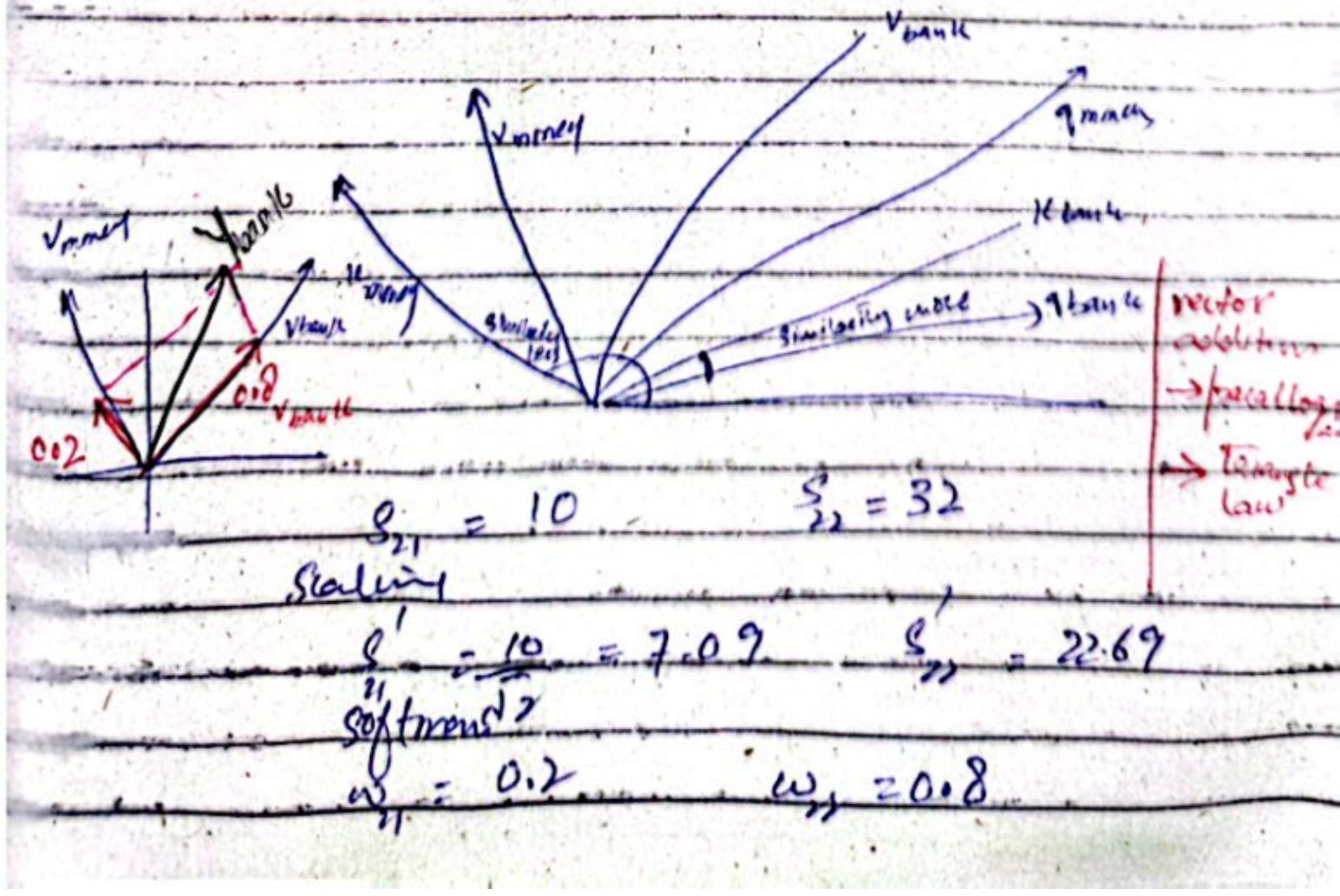
$$e_{bank} \cdot w_v = [9 \;\; 3] \begin{bmatrix} 4 & 1 \\ 2 & 1 \end{bmatrix} = [36+6 \;\; 9+2] = [42 \;\; 11]$$



$$s_{bank} \cdot k_{money}^{T} = [2 \;\; 7] \begin{bmatrix} 31 \\ 15 \end{bmatrix}$$

$$[82+95] = [-]$$

q_{bank}



$s_{21} = 10$ $\quad s_{22} = 32$

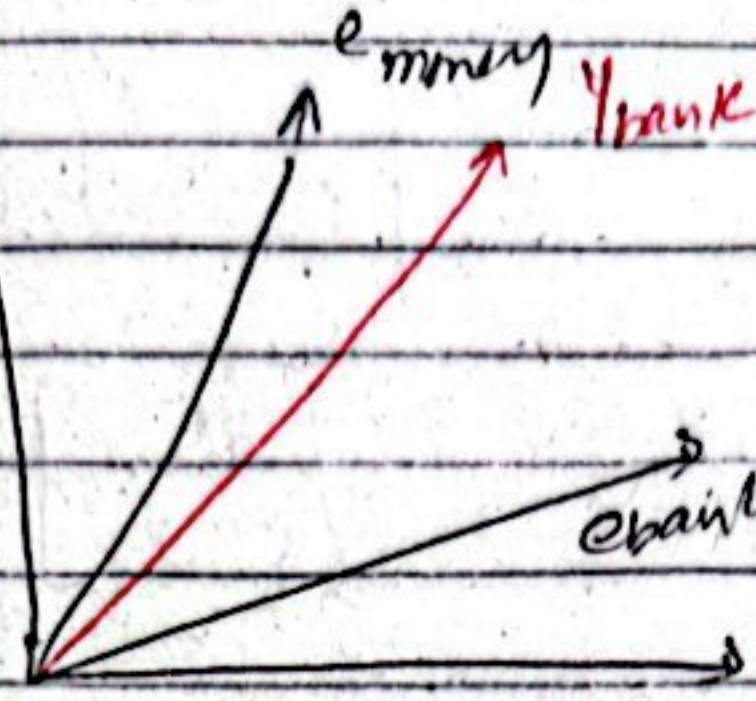
Scaling

$s'_{11} = \frac{10}{\sqrt{2}} = 7.07$ $\quad s'_{22} = 22.67$

"Softmax"

$w_{11} = 0.2$ $\quad w_{22} = 0.8$

money bank



bank embedding is too far when money embedding

When we apply self attention then Y bank contextual embedding is too closest to money embedding

Self attention on the basis of content embedding generate

Self attention acts like gravity

money word pull bank towards ourself.

self attention
self is context aware.

Why Self attention called "Self"?

Why self attention called "attention"?

Recap of Attention :-

language translation task solve using rnn n/w.
⇒ that type of task (sequence to sequence)
used Encoder Decoder Architecture.

↳ Encoder :- feed sentence word by word, and
encoder process sentence and it maintain hidden
state and use the end hidden state
are (context vector) and that
context vector send to decoder.
⇒ context vector contain summary of
sentence.

Decoder pick context vector and step
by step passing output. print.

This approach is good. but one big
problem in it.
you whole put burden on context
vector. (context vector is nothing but
just a set of numbers)

but in some way your sentence size is
greater than 30. in that case, you
can't effectively summerize your sentence
in context vector.

then researcher notice sentence >30,
translation quality bad. Then

attention mechanism came.

Attention :-

Instead
after seeing whole sentence, its summary only put in one vector and send it of decoder ~~if~~ you.

~~if~~ decoder each timestep, if you send different different context vector and what context vector has info in which input words is useful to print that word.

~~use~~ with the help of hidden state we make that context vector

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j$$

α tell ~~what~~ this current ~~output~~ word which input hidden state is useful

How much - total alphas
= total input word * - Total output words

How α are calculate ?

$$\alpha_{ij} = \text{Softmax}(e_{ij})$$

alignment score

Using attention :-

$e_{ij} = s_i h_j$

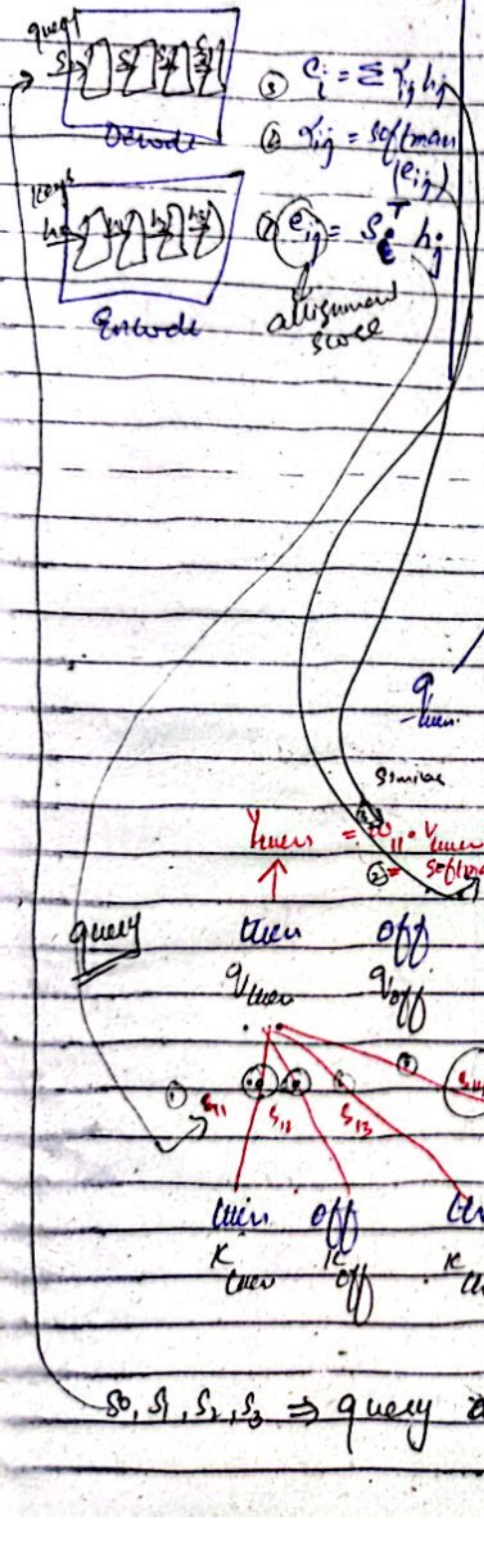
weights

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

$\alpha_{ij} = \text{softmax}(e_{ij})$

$e_{ij} = s_i^T h_j$

Self attention



that's self attention called attention.

Long attention find the two different sequence
↓
alignment score

inter sequence
eg
↓
cross

Self attention (same sequence)
↓
similarity score

Intra sequence
eg
↓
eg

→ multihead
→ position encoding
→ transformer architecture

Multi-head Attention :-

Recap of self-attention :-

Self-attention is a technique which you used to generate contextual embedding

money bank → represent number

1- One hot encoding
2- BOW
3- TF-IDF
4- Embedding (semantic meaning capture)

~~money bank~~
⇒ problem
⇒ Static
e.g ↳ money bank
↳ river bank
both sentence bank embedding is same but its meaning is different in two sentence.

To solve that problem.

5- Self embedding came
↳ generate contextual embedding
(which context it came, it change)

Problem in self attention

The man saw the astronomer with a telescope

⇒ two perspective of sentence
ایک آدمی نے Telescope کے ساتھ astronomer کو دیکھا
یا Telescope والے astronomer کو آدمی نے دیکھا

This sentence is ambigious.
but unfortunately self attention only capture one meanings/perspective
Self attention can't capture multiple perspective.

working of self attention
↳ embedding of each word find
↳ query, key, value find of each word.
↳ then similarity check b/w query, key, value vector

1) possible
high similarity score
man , telescope ←
saw , " ←

2) saw meaning
astronel , telescope ↙
saw , astronel ↙

doc summarization
→ If paragraph based on self attention it generate one perspective summary
we want multiple perspective summary.

similarity score

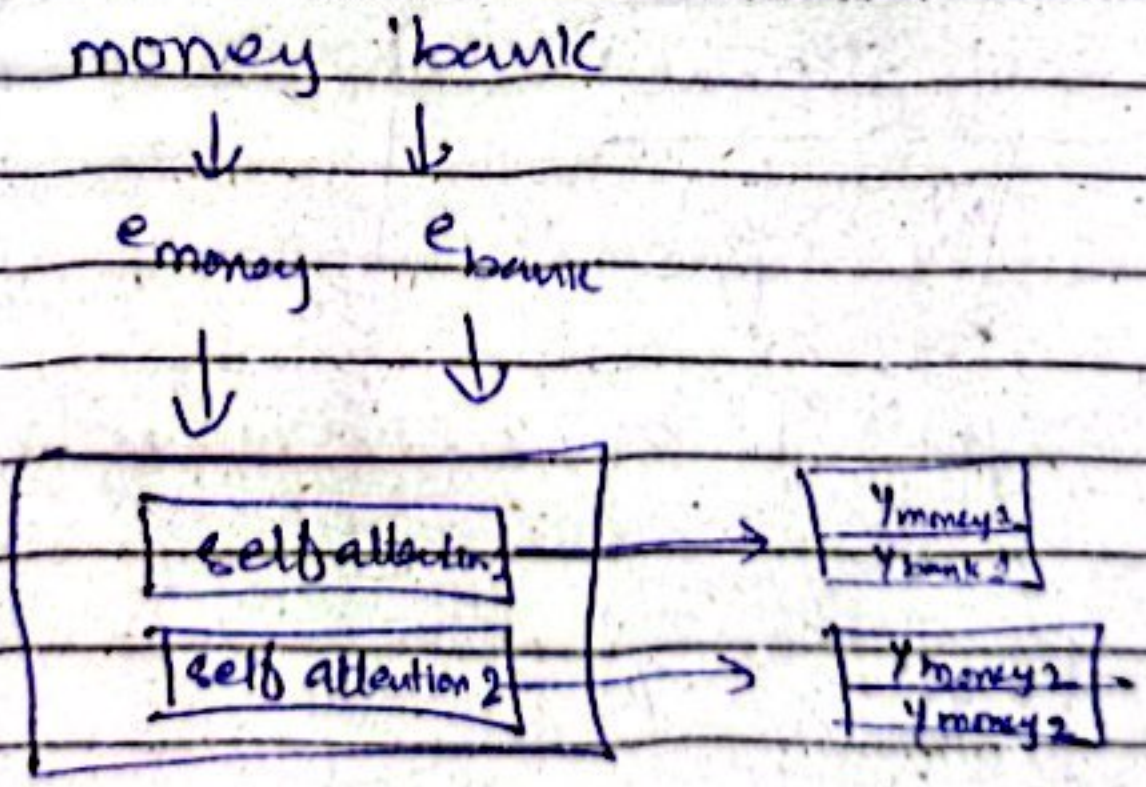
	The	man	eats	the	apple	with	a	telescope
The								
man								
eats								
the								
apple								
with								
a								
telescope								

Self attention only make one table.
multiple meaning / perspective can't capture.

To solve this problem multi-head attention came.

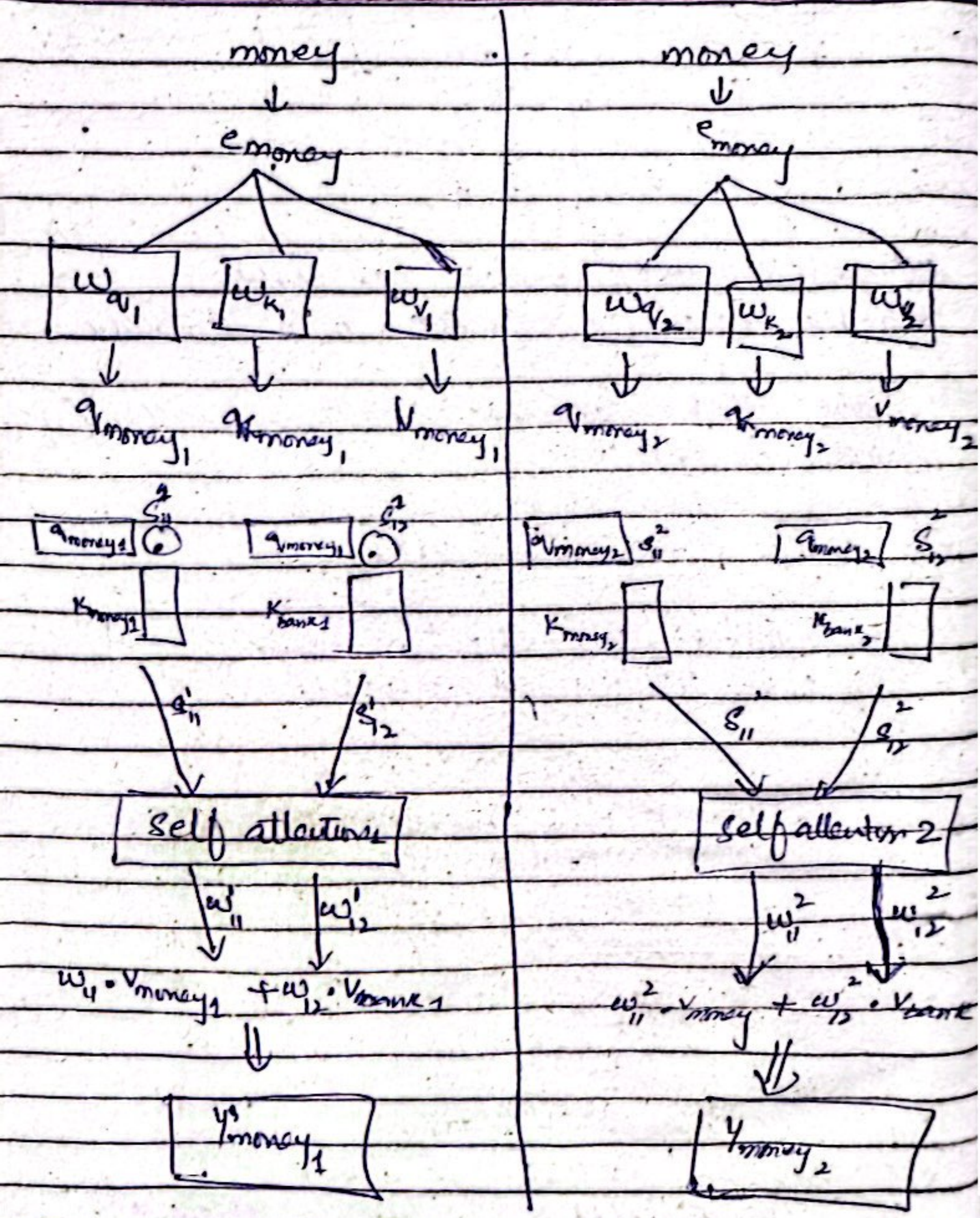
Multi-head attention

⇒ multiple self-attention used are Multi-head attention

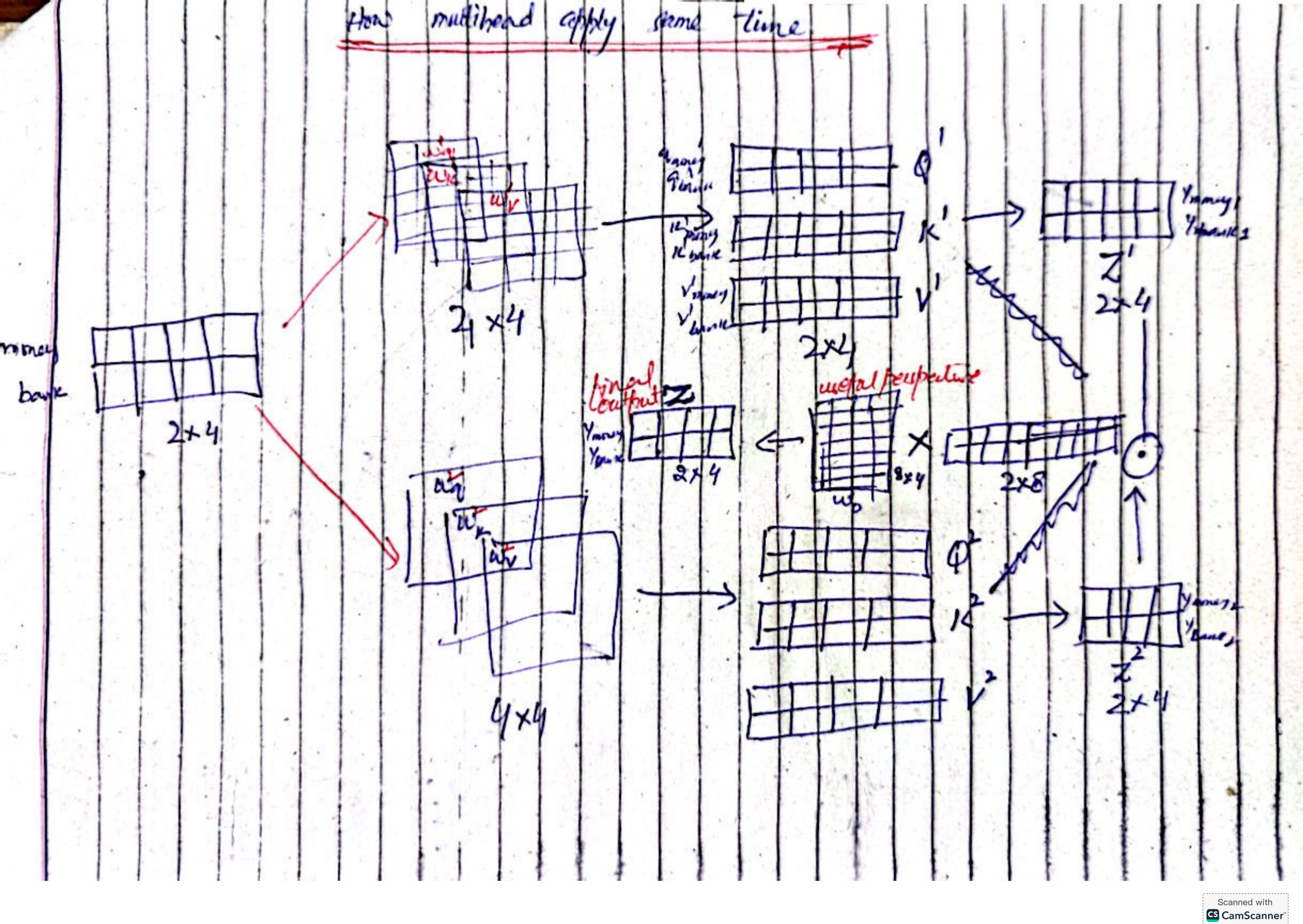


It parallely do, capture different perspective.

Sem 1 - money bank

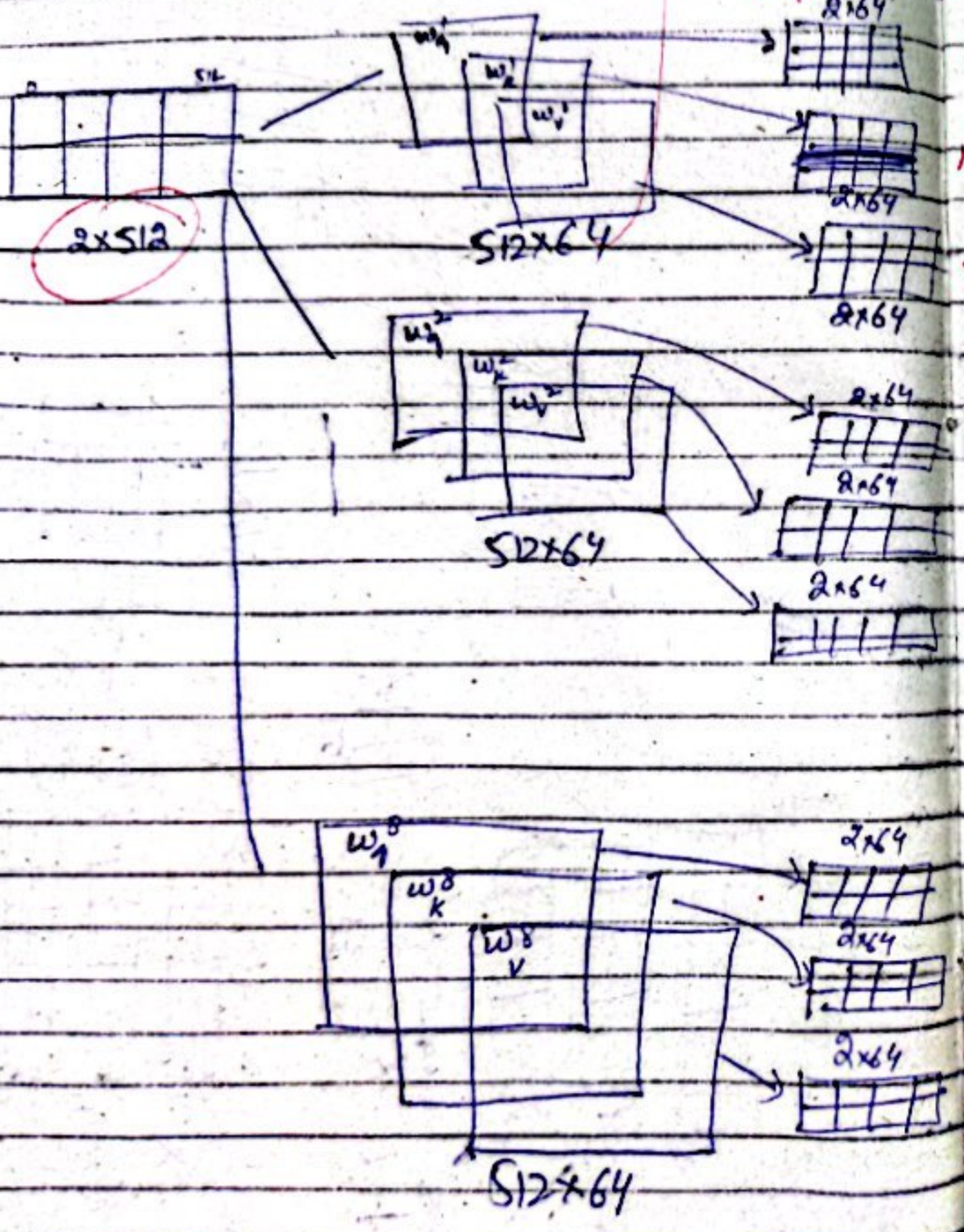


How multihead apply same time



Total 24 Million

Why we go 64? Because of doing computation less.



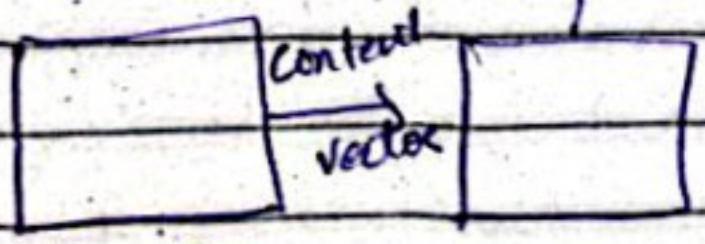
Masked Multi head Attention

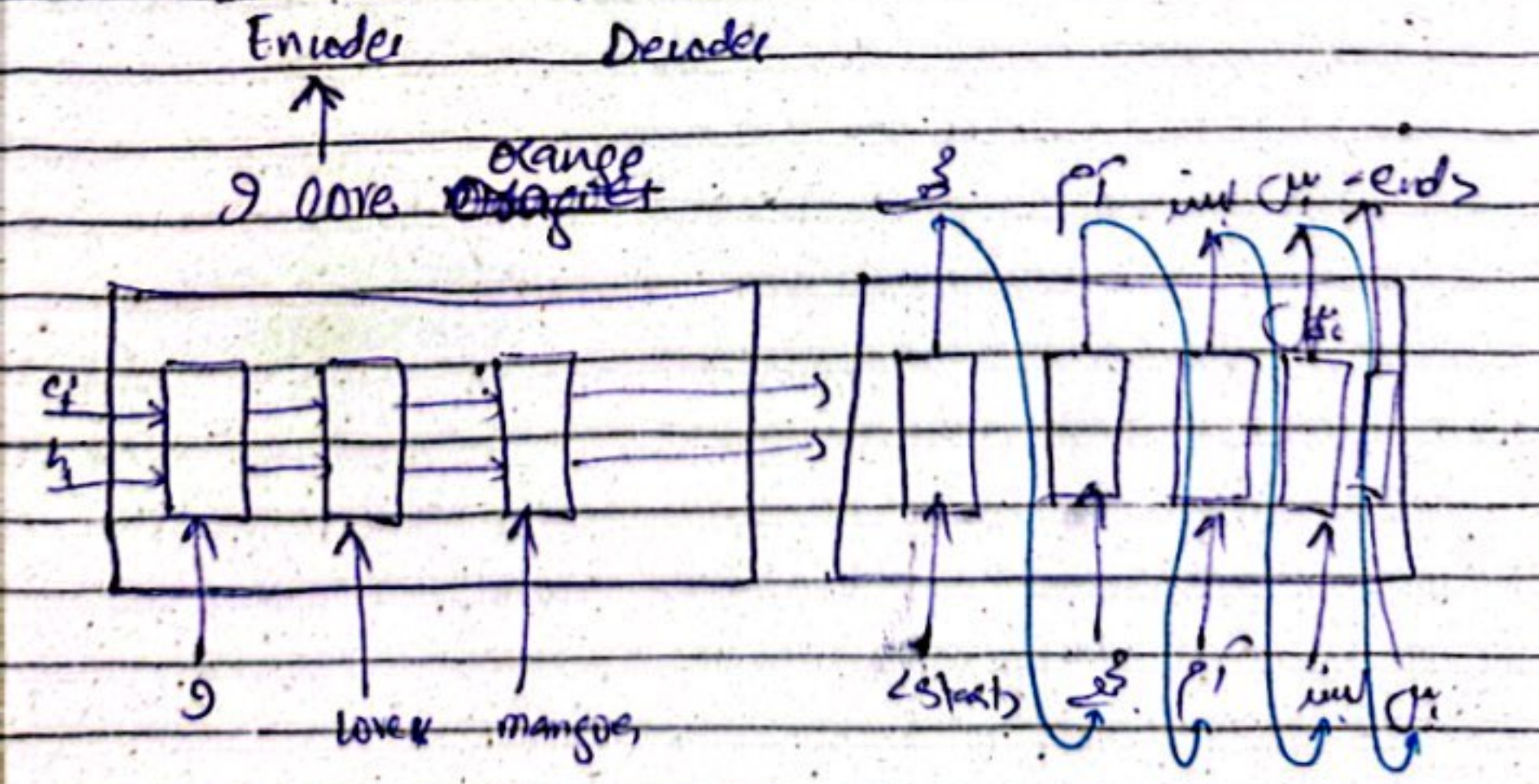
The transformer decoder is autoregressive at inference time (prediction / testing) and non-autoregressive at training time.

Autoregressive ⇒

Autoregressive model are a class of models that generate new data points in a sequence by conditioning each new point on the previous generated points.

next prediction based/depend on previous sequence





time step what output.

at any time-step, if you predict output you need previous

you called that model called autoregressive.

Why seq to seq are autoregressive?
(when we predict next word you need to know what is predict before.
because next word based on previous words.

That's why we can't do that thing is parallel.

⇒ Transformer in training time not auto-regressive.

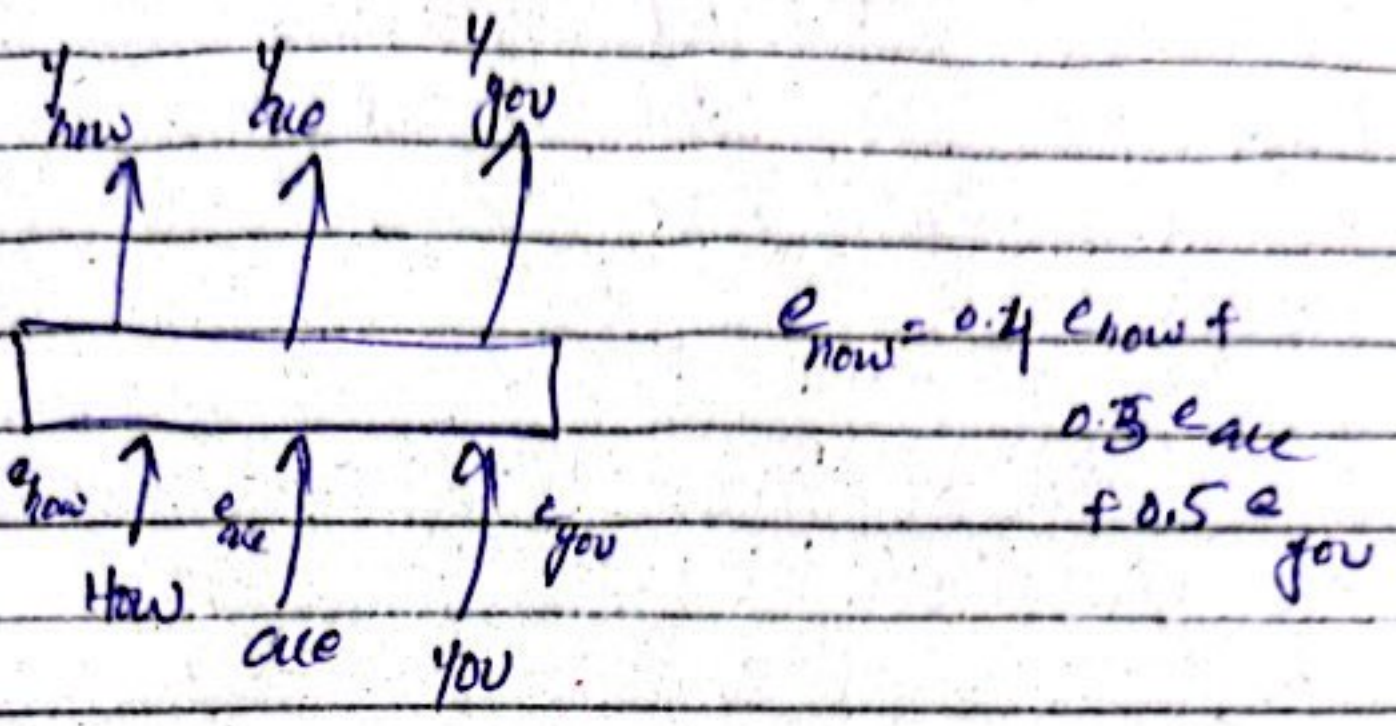
if in training ⇒ we used teacher Enforcing. we don't give wrong word in training. ~~the~~
in training, we know the output of sentence. why we not only give the output ~~the~~ of new word from dataset without seeing what is produced.

⇒ Is there any drawback of making autoregressive? (training)
If 300 words, its means 301 times seperate Run decoder.

→ You do that thing in parallel way also during training because you know the next word from dataset you.

Self attention ⇒ next word contextual embedding

you know the next word.



in teiling e_{how} will make from future prediction word value and ownself also.

⇒ you don't know the next word so to solve that eq.

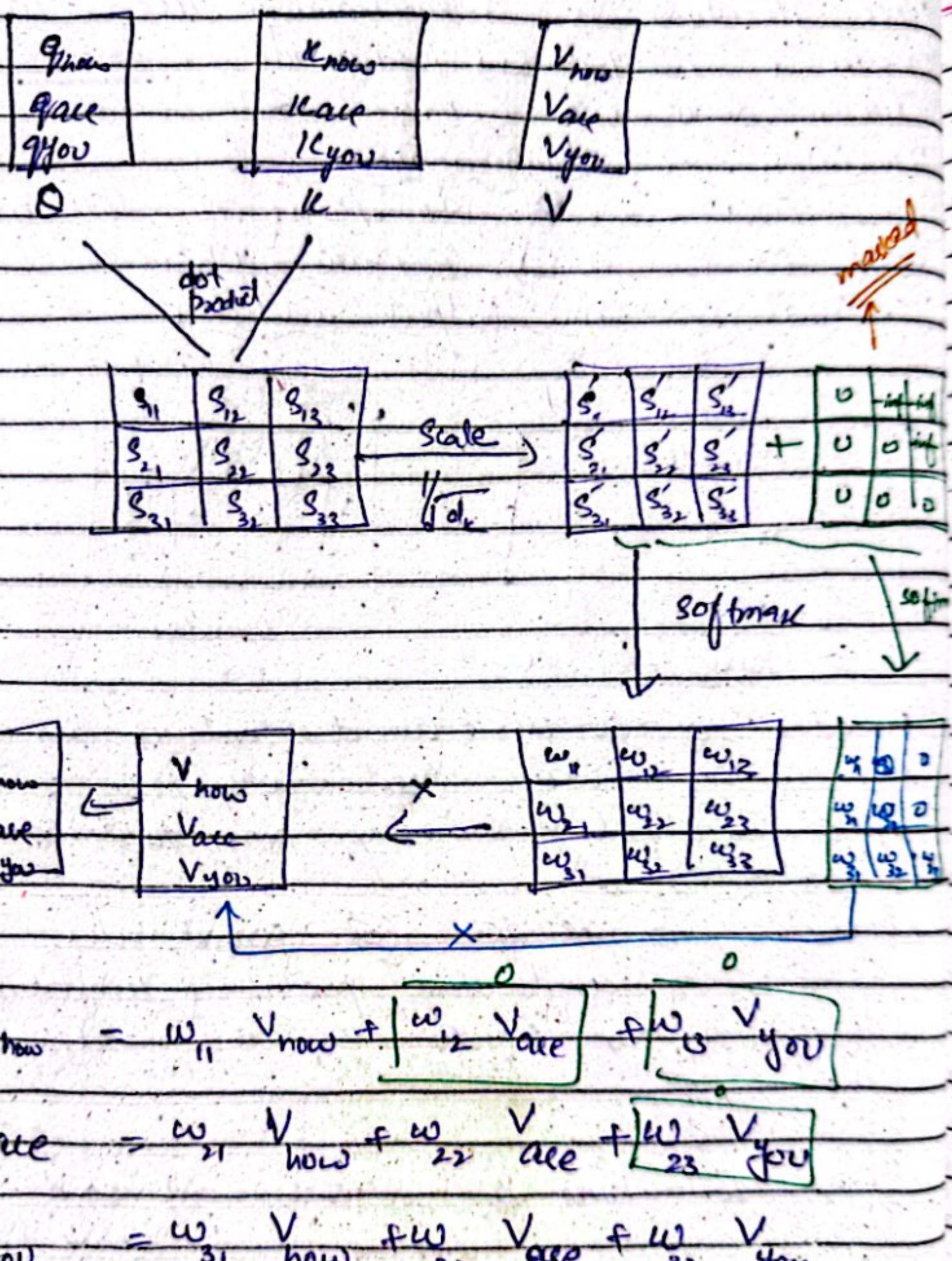
$e_{how} = 0.2\, e_{how} + 0.1\, e_{are} + 0.2\, e_{you}$

$e_{are} = 0.3\, e_{how} + 0.6\, e_{are} + 0.2\, e_{you}$

$e_{you} = 0.7\, e_{how} + 0.3\, e_{are} + 0.2\, e_{you}$

Autoregressive
- no data leakage
- slow

Non-Autoregressive
- fast
- data leakage



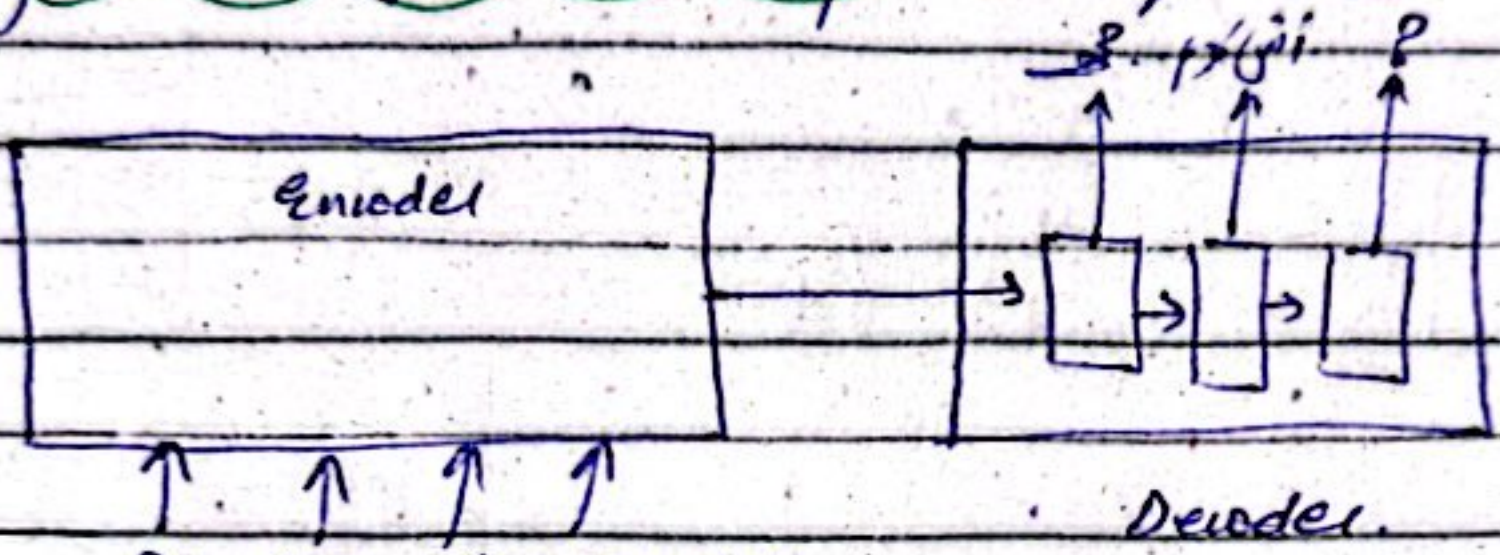
$$y_{how} = w_{11} v_{how} + w_{12} v_{are} + w_{13} v_{you}$$

$$y_{are} = w_{21} v_{how} + w_{22} v_{are} + w_{23} v_{you}$$

$$y_{you} = w_{31} v_{how} + w_{32} v_{are} + w_{33} v_{you}$$

Cross Attention

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



Encoder ⇒ to understand sentence in very good way
that summary pass to decoder.

Decoder ⇒ decoder get context vector and start predicting/translation
⇒ seq to seq it predict/translate

Decoder time 3 ⇒ what next word ↓ depend

Self attention (same sentence context generate)

get from

decoder what generate until till now

input sentence (encoder information

what is relationship b/w input and output sentence.

	میں	آئس کریم	کھانا	پسند	ہے
I					
like					
eating					
ice cream					

different
two sequence what relationship

↓

Cross self attention

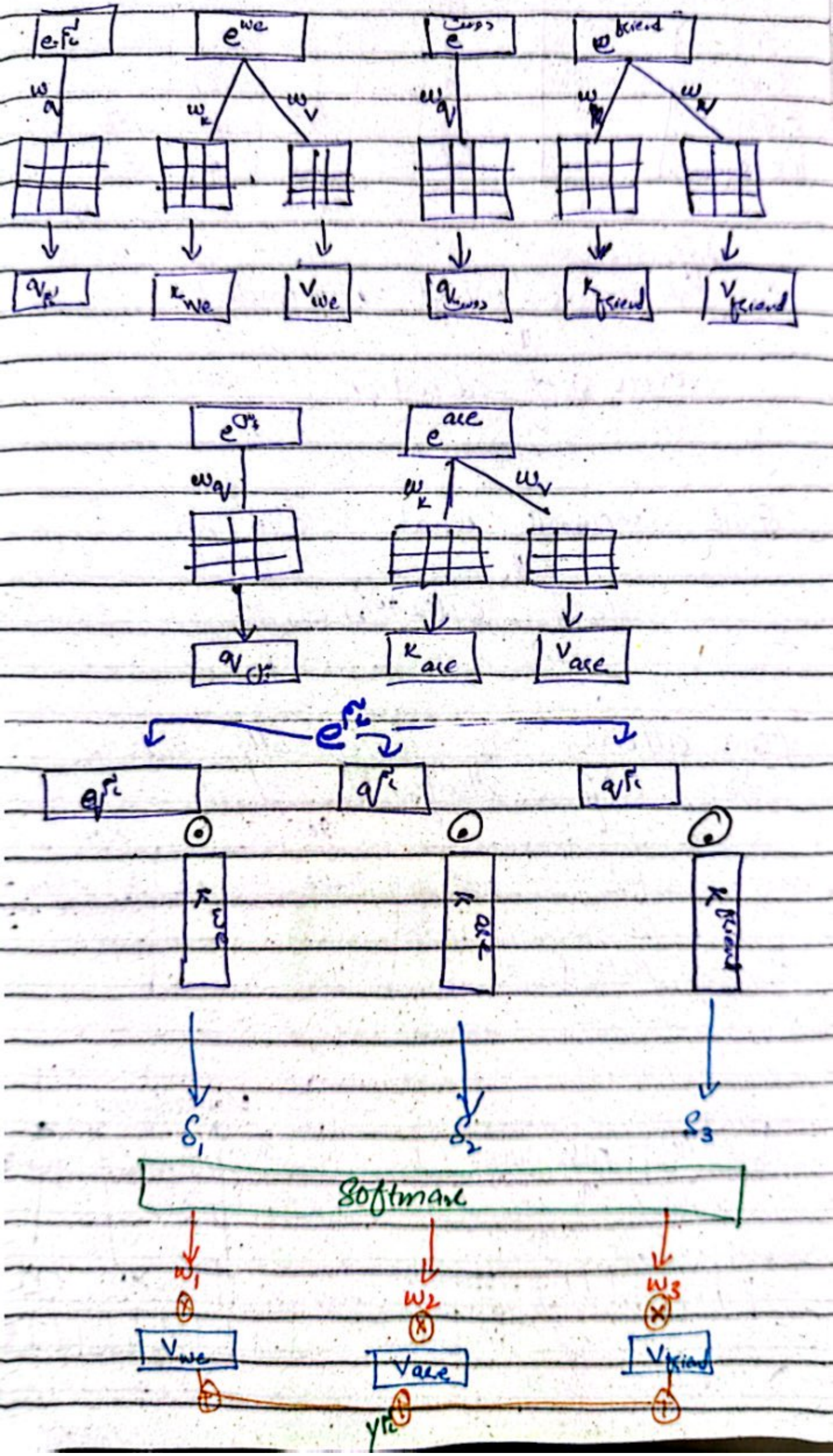
Same sequence what relationship

↓

Self attention

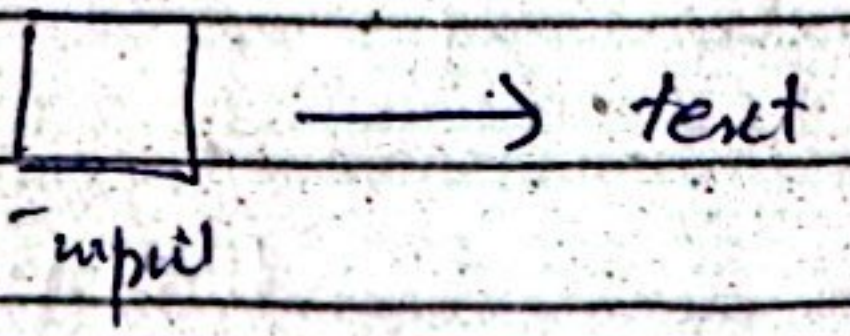
Cross attention is conceptually similar to self attention

	Self	Cross
→ input	Single sequence embedding	two different sequence embedding
→ processing		
→ output		


Softmax
Scanned with CamScanner

Use cases

- image captioning



- text → image
- text → speech