

Transformers

What is Transformer?

Defⁿ:- It is a neural-network architecture which helps to handle "seq 2 seq" tasks.

Seq2Seq Task:- The task in which both input and output is text.

Examples:-

- i) Machine translation
- ii) Question answering System
- iii) Text Summarization
- ⋮
- etc

Impact of Transformers:

next page ...

Impact of Transformers

Revolution in NLP

50 years

- heuristics
- Statistical ML models
- embeddings
- RNNs
- LSTMs

But with Transformers we have "State-of-the-art" (SOTA) NLP technique

Democratizing AI

Since they can be Scalable for huge data large models were built like "BERT", "GPT" and which are accessible for everyone!

Multimodality capability

Not only text as input we can also give image, audio, video and also generate them.

Eg: like DALL-E, Chat-GPT, Midjourney, Runway-ML

Acceleration of Gen AI

To generate images before GANs were used not SOTA but with transformers

different applications built like

"DALL-E" - generate images and Gen AI growth increased!

Unification of deep learning

for different deep learning problems we are only using Transformers before we used ANN for text CNN for images and RNN for text

Gen AI CV, NLP seq2seq Reinforcement

The Timeline:

2010's :- 2014 : Used RNNs / LSTMs, Attention mechanism introduced

2017 - Present: The era of Transformers and Advanced AI

2017: Transformer architecture introduced

2018: BERT and GPT-x Models developed.

2020: Introduction of Transformers

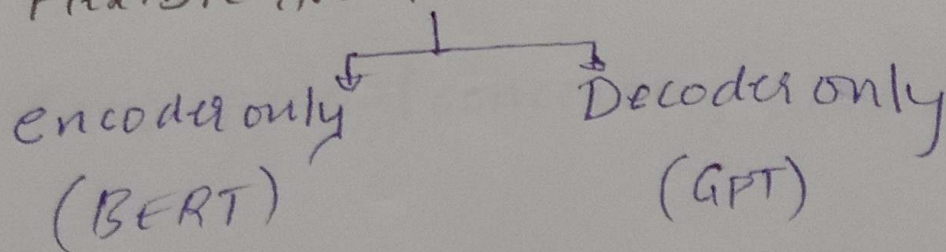
2021: AlphaFold-2, Tools like DALL-E

2022 - Now: ChatGPT, Gemini & other language models introduced!

* Continued Growth ... potential moves towards AGI.

Advantages:

1. Scalability
2. Transfer learning
3. Multimodal input and output
4. Flexible Architecture



5. ecosystem: It has vibrant community and libraries like "Huggingface", and a lot of content is available like videos, blogs - etc

6. Integrated AI Techniques:

1. DALL-E - GANs + Transformers
2. Reinforcement Learning + Transformers
3. Vision Transformer - CNN + Transformer
4. GNNs + Transformers ..

Disadvantages:

1. High computation: Lot of GPU's required which are very costly.
2. Data: Lot of data is required, when we are doing supervised ML problems we require lot of labeled data.
3. Overfitting
4. Energy consumption: Lot of electricity required
5. Interpretability: We get SOTA Results, but we can't interpret how we are achieving
6. Bias: Ethical concerns, hallucinations

What is Self-Attention:

Defⁿ:- It is a mechanism that can take Static Embeddings as input, and can generate good contextual embeddings, which are much better to use any kind of NLP Task.

What are static Embeddings?

- 1) I love Apple fruit
- 2) Apple is healthy
- 3) Apple is better than orange
- 4) Apple makes great phones

⋮

1000

* Assume in above 1000 documents, 900 Apple is a fruit and in 100 document 'Apple' is a phone.

Fruit Phone
↑ ↑

The Embedding of Apple = $[0.8 \ 0.2]$, when we use techniques like above, Word2Vec etc

* Now we have created embeddings of each word in vocabulary using wordvec.

* Assume I want to convert a "English" sentence to "Hindi". Nothing but Machine translation task

* Assume following Test document:

Apple launched a new phone while I was eating an orange

→ Now w.r.t our embedding (which are static) Apple will be fruit since embedding of

Apple = $[0.8 \quad 0.2]$ (probability of Fruit is high)
 ↓ ↓
 Fruit Tech

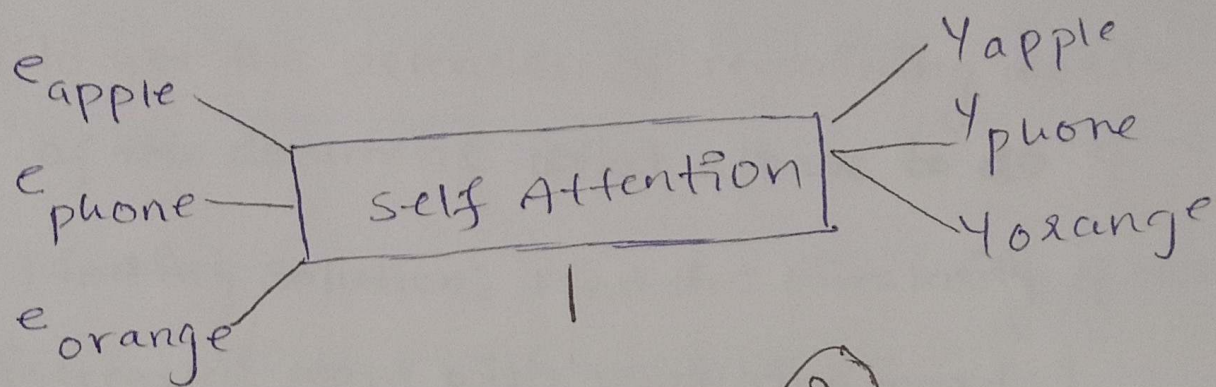
But

Apple is not a fruit, It is brand w.r.t our Test document

We are in a problem

Solⁿ:- Self Attention

* Using Self-Attention creates new embedding for apple dynamically according given Test document!



How Self-Attention works:

Assume only 2 documents are there in dataset

1. Money bank grows
2. River bank flows

* I am using my mind and writing:

$$e_{bank} = 0.2 e_{money} + 0.7 e_{bank} + 0.1 e_{grows} \quad (1)$$

e_{bank} , e_{money} , e_{grows} are the "embeddings"

Vectors created using techniques like glove, word2vec

... So on.

Eg (1) says e_{bank} is not only e_{bank} created by wordvec but it also depends on words in the same document here: e_{money} , e_{grows}

* To get the attention of remaining words of the document what I have to do ?

* ~~Finding~~ Solution: Find the similarity of the interested word with remaining words!

* How to find similarity ?

Soln': Using dot product

* How dot-product will give use similarity ?

Soln': We know that similar words will have nearly same embedding vector when we use word2vec or Glove techniques

$$\text{For ex: } e_{\text{bank}} = [5 \ 4 \ 2]$$

$$e_{\text{money}} = [5 \ 3 \ 1]$$

$$e_{\text{orange}} = [2 \ 1 \ 1]$$

$$\begin{aligned} e_{\text{bank}} \cdot [e_{\text{money}}]^T &= [5 \ 4 \ 2] \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix} \\ &= [25 + 12 + 2] = 39, \end{aligned}$$

$$e_{\text{bank}} \cdot [e_{\text{orange}}]^T = 10 + 4 + 2 = 16,$$

Since bank, money comes to similar category

similarity is higher (39), And there is no relation between bank and orange so that similarity is less (16) compared to money!

* I hope you understood how dot product helps to find similarity.

* Now let's come to our example:

money bank grows

$e_{\text{bank}} \cdot [e_{\text{money}}]^T$ will give you how much they are similar. Since we want the vector let's multiply this scalar with e_{money} .

$\Rightarrow$ It will become:

* $(e_{\text{bank}} \cdot [e_{\text{money}}]^T) e_{\text{money}} \rightarrow$ It is a vector which tells how much e_{money} is similar to e_{bank}

* Now I write e_{bank} with other words attention as:

$$e_{\text{bank}} = e_{\text{bank}} \cdot [e_{\text{money}}]^T e_{\text{money}} + e_{\text{bank}} [e_{\text{bank}}]^T e_{\text{bank}} + e_{\text{bank}} \cdot [e_{\text{grows}}] e_{\text{grows}} \quad - (2)$$

$$e_{\text{bank}} = 0.2 e_{\text{money}} + 0.7 e_{\text{bank}} + 0.1 e_{\text{grows}} \quad \text{--- (1)}$$

which I wrote at the start

Can you find the similarity b/w ① & ②?

$$\begin{aligned} e_{\text{bank}} \cdot [e_{\text{money}}]^T &= 0.2 \\ e_{\text{bank}} \cdot [e_{\text{bank}}]^T &= 0.7 \\ e_{\text{bank}} \cdot [e_{\text{grows}}]^T &= 0.1 \end{aligned} \quad \left. \vphantom{\begin{aligned} e_{\text{bank}} \cdot [e_{\text{money}}]^T &= 0.2 \\ e_{\text{bank}} \cdot [e_{\text{bank}}]^T &= 0.7 \\ e_{\text{bank}} \cdot [e_{\text{grows}}]^T &= 0.1 \end{aligned}} \right\} \text{are the values of similarity}$$

How what if the Dot-product is not between 0-1?

Soln:- We use softmax function

* Let's say

$$e_{\text{bank}} \cdot [e_{\text{money}}]^T = s_{21}$$

$$e_{\text{bank}} \cdot [e_{\text{bank}}]^T = s_{22}$$

$$e_{\text{bank}} \cdot [e_{\text{grows}}]^T = s_{23}$$

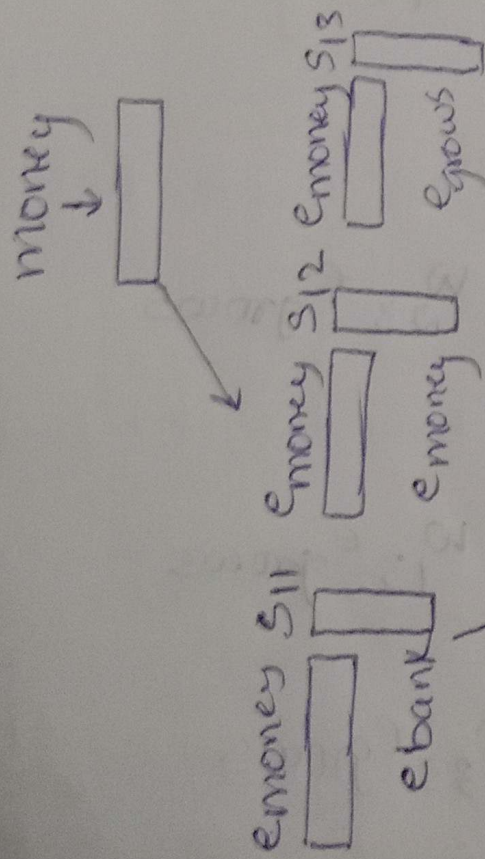
$$w_{21} = \frac{e^{s_{21}}}{e^{s_{22}} + e^{s_{21}} + e^{s_{23}}} \quad w_{22} = \frac{e^{s_{22}}}{e^{s_{21}} + e^{s_{22}} + e^{s_{23}}} \quad \text{|| by } w_{23}$$

Now e_{bank} becomes

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

$$\text{|| by } e_{\text{money}} = w_{11} e_{\text{money}} + w_{12} e_{\text{bank}} + w_{13} e_{\text{grows}}$$

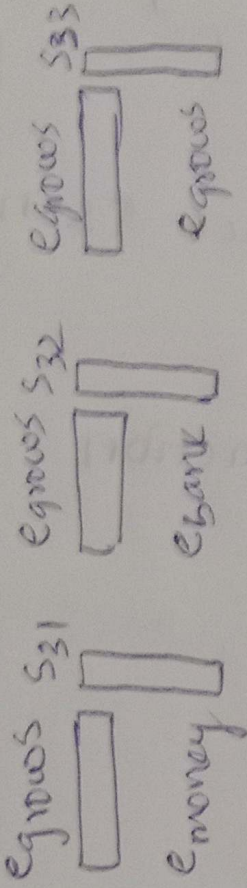
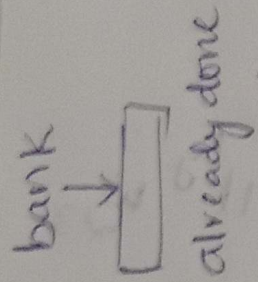
$$e_{\text{grows}} = w_{31} e_{\text{money}} + w_{32} e_{\text{bank}} + w_{33} e_{\text{grows}}$$



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

||

easy money

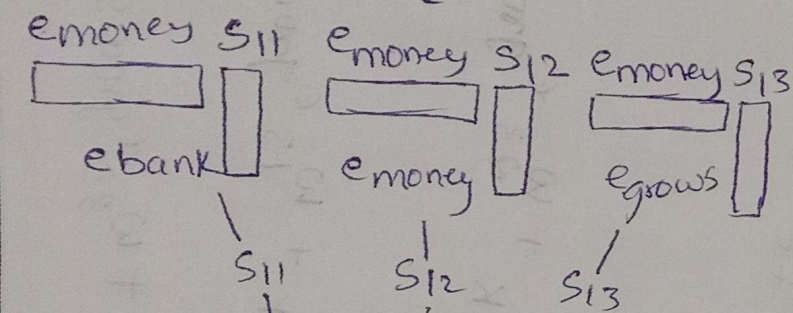
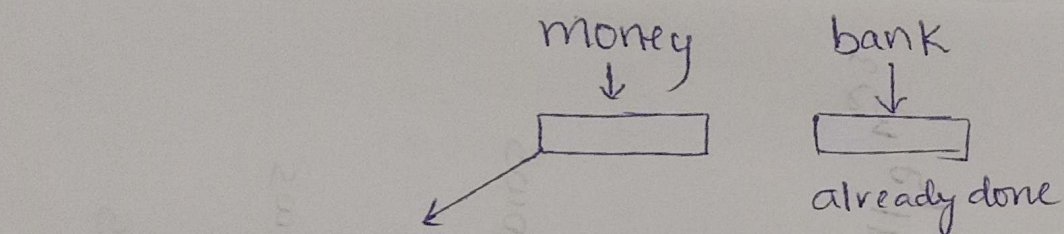


$$w_{31} \times e_{\text{money}} + w_{32} \times e_{\text{bank}} + w_{33} \times e_{\text{grows}} =$$

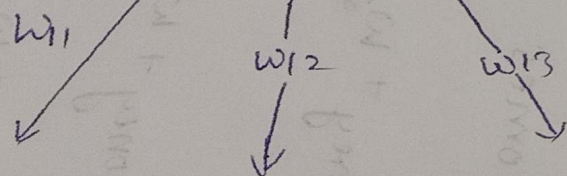
||

easy grows

Summary

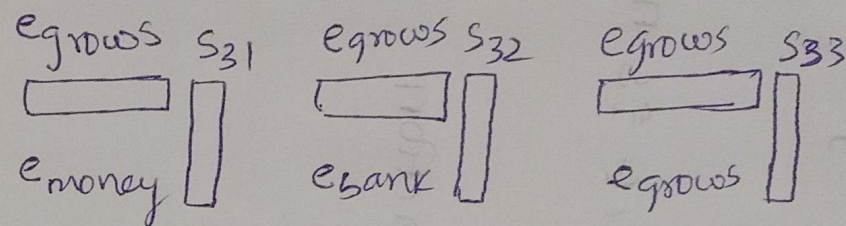


Softmax

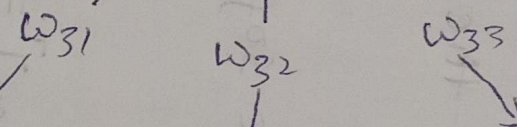


$$w_{11} \times e_{\text{bank}} + w_{12} \times e_{\text{money}} + w_{13} \times e_{\text{grows}}$$

||
e_{or} y
money



SOFTMAX



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

||
e_{or} y
grows

Summary

Congrats!! , Now you know how self Attention works!

What is the Advantage of this Approach?

we have created a very good embeddings of initial embeddings

→ This Approach is parallel!

I think you might have observed that to create new context embedding of one word or token we just need other words embedding which we created by word2vec!

* We can create new embeddings at the same time!!

→ In this approach there are no parameters involved!

* which implies there is no learning from the data! Again the created new embedding become static but has within the document only!! context

* The new embedding which we created can capture the context of the document only. It doesn't matter what is the label of the document!

* But we have to create embedding which is "Task-specific" I mean which also captures the context of label

Why we need Task-Specific embedding?

Let me explain with example!

Let's say one of the document is the following and we are dealing with Machine Translation task.

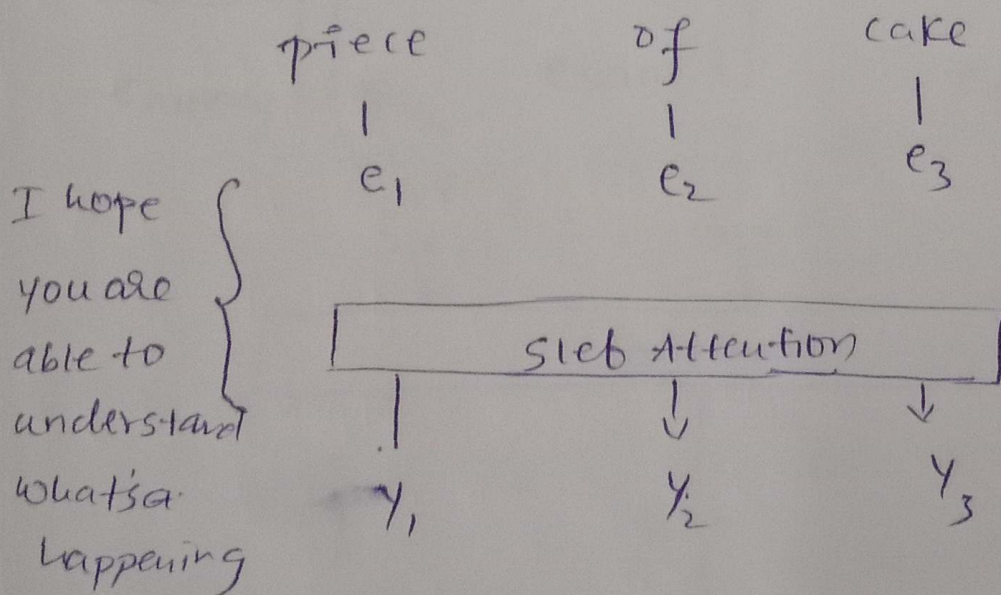
document

label

piece of cake

$y_t = \text{कैक का टुकड़ा}$

→ Lets try to predict hindi translation of



with $[y_1, y_2, y_3]$ model can predict this

$y_p = \text{कैक का टुकड़ा}$

* As you can see both y_t and y_p are completely different!

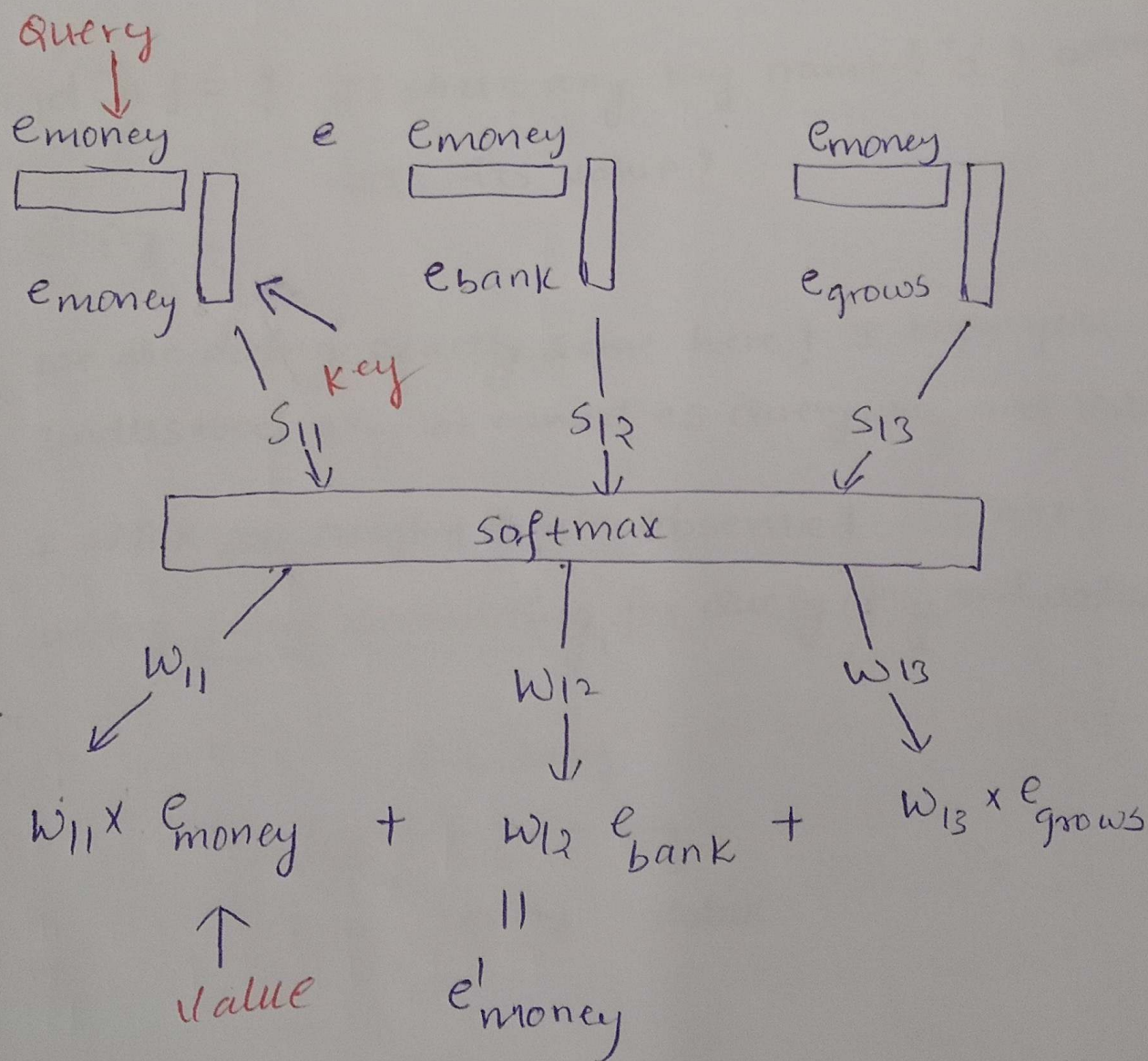
How to find Task-Specific Embeddings?

Solⁿ: Now we have to introduce some learning parameters which can learn from data! and help us to update our new embeddings!

Now Query, key and Value Matrices comes to

play:

Lets observe this:



Why Query?

I named it because the e_{money} is asking or searching or querying other embedding that "how much I am similar to you".

Why keys? Why values?

It is basic computer science. ^{value} ^{value} ^{value}

It is basic computer science.

Let's have a dictionary: $\{a:2, b:3, c:4\} = d$

Diagram illustrating the dictionary structure:

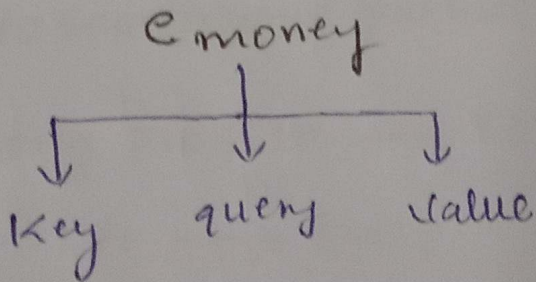
- Key: a maps to Value: 2
- Key: b maps to Value: 3
- Key: c maps to Value: 4

The entire dictionary is labeled d .

$d[a] = ?$ Is there any key named 'a'? what
 its value?

Query

- * we are doing exactly same here! I hope you understood why we named as Query, Key and Value.
- * I think you might have observed we are using same embedding for query, key and value.



* In this process of determining contextual embedding every embedding vector playing three roles?

- Query
- Key
- Value

The problem?

The problem is same embedding vector for

example $e_{\text{money}} = \text{Query}$

$e_{\text{money}} = \text{Key}$

$e_{\text{money}} = \text{Value}$

Ideally we have to derive query, key and values from e_{money} !

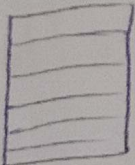
what do you think?

Analogy:- Lets say you have written a book on yourself (autobiography) and became a successful Author, now your ready for marriage! you went to search your partner in "JeeVansathi"

1) Strong profile $\rightarrow$ Key

2) search $\rightarrow$ Query

3) matching partner $\rightarrow$ Value

*  = e_{bank} (assume)
your book

What if you not built profile?

If someone want to know your highlights
"you will say them to read complete book?"

no right!

Now your searching for someone!

would you give your complete book on to the
search bar (because you have written in book
which partner you want) to search partner?

* you have to provide crisp information to
get good results on search right!

** In the similar way we should not use same
embedding vector for query, key and value

* Matching vector is really good when you
are meeting someone!

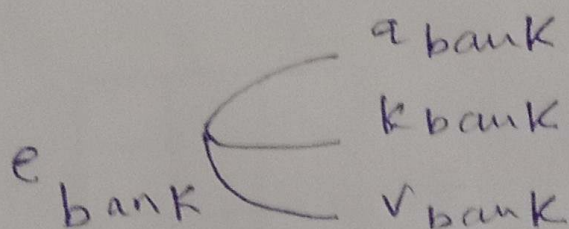
* I hope you got the solid understanding why
we should create query, key and values

* Now, let's say initially you have profile says I am the author of the book [name] (political) now they started recommend your profile for those who interested in politics! But you thought "they always talk about politics I don't want them". Then you updated your profile!

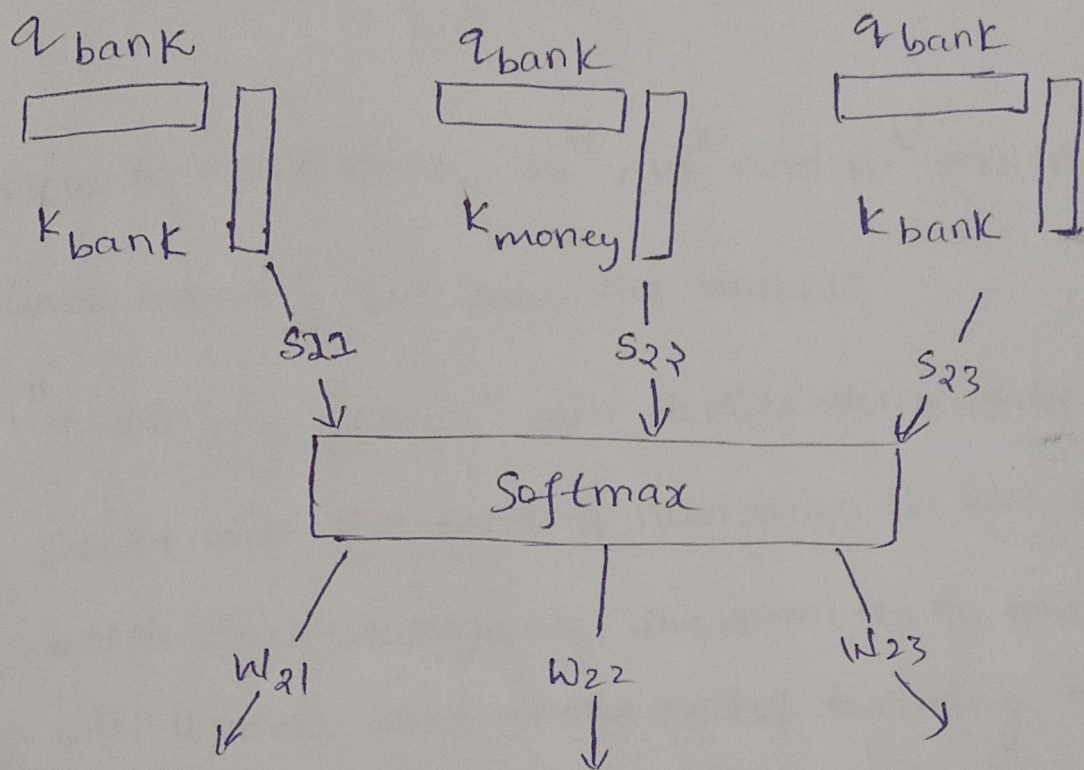
* What you did is you have updated your profile based on **DATA**. Your understanding the data and updating your profile, changing search pattern and meeting with people

* In the similar way, q_{bank} , k_{bank} and v_{bank} should learn according to data!

remember! We want the same! we have to modify our new embedding according to the task (DATA)



Finally Contextual Embedding Vector created as:
 Lets create the e_{bank}



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

How to Find Query, Key and Value Vectors

Solⁿ: Using Linear Transformations

$$e_{bank} \cdot \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix} = q_{bank}$$

(W^q)

$$e_{bank} \cdot \begin{pmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{pmatrix} = k_{bank}$$

(W^k)

$$e_{\text{bank}} \cdot \begin{matrix} \boxed{\begin{matrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{matrix}} = v_{\text{bank}} \\ \text{\textcolor{red}{W}^V} \end{matrix}$$

Initially random number

How to Find these W^Q , W^K and W^V matrices?

Guess what? we know the answer

- * "Training process" will decide the numbers. we start with the random numbers in the matrices!
- * With Backpropagation the numbers in matrices will update and at the end of training we will be having good W^Q , W^K and W^V matrices.
- * In the similar way we calculate

$$e_{\text{money}} \cdot \boxed{W^Q} = q_{\text{money}}$$

same as before

$$e_{\text{money}} \cdot \boxed{W^K} = k_{\text{money}}$$

same as before

$$e_{\text{money}} \cdot \boxed{W^V} = v_{\text{money}}$$

same matrix which used to calculate v_{bank}

Small y , Small q , Small K and Small V

finally (congrats) 😊

