Understanding Q,K,V In Transformer(Self Attention)

mustafac · Follow
Published in Analytics Vidhya · 8 min read · Jan 1, 2021

I am writing this article as a follow up of previous articles on Transformer.(Link) Here I assume you already know what is Transformer and just want to understand more about Q,K,V vectors in the architecture.

I get the base code from here . But I changed so much that you cannot match the lines probably. The code is at github, but use nbviewer.com to view to see colors.
Code link in nbviewer
Code link in github

Let me over simplify everything. What is a vector in **n** dimension?
It is encoding data with **n** features. So when dimension is high it is hard to visualize.

So lets do simplest thing we can.
Let's say we have below sentences is :

eat apple
eat bread
drink water
drink beer
read newspaper
read book

	Food realeted	Reading Related
eat apple	0.8	0.1
eat bread	0.7	0.2
drink water	0.7	0.1
drink beer	0.7	0.1
read newsp	0.1	0.8
read book	0.2	0.8

Vector table for dimension 2

If we generate 2 dimensional vector for these sentences(not a vector for word, vector for sentence) , we can create as above.(The more dimension, better you encode) With this over(over , over) simplification if we multiply these sentence vector we see that food related gets more point when multiplied with food related items.

eat apple x eat bread = 0.8 * 0.7 + 0.1 * 0.2 = 0.58
eat apple x read book = 0.8 * 0.2 + 0.1 * 0.8 = 0.24

This was dimension 2. Now think that how many dimensions we actually need for an NLP problem. Back to our problem, our problem is learning English-German translation. And our vectors are 64 dimensional. I will try to reduce them dim 2 or 3.

In transformer **Q,K,V** are vectors we use to get better encoding for both our source and target words.

Q: Vector(Linear layer output) related with what we encode(output, it can be output of encoder layer or decoder layer)
K: Vector(Linear layer output) related with what we use as input to output.
V: Learned vector(Linear layer output) as a result of calculations, related with input

In Transformer we have 3 place to use self-attention so we have Q,K,V vectors.

1- **Encoder Self attention**
Q = K = V = Our source sentence(English)

2- **Decoder Self attention**
Q = K = V = Our target sentence(German)

3- **Decoder-Encoder attention**
Q = Our target sentence(German)
K = V = Our source sentence(English)

With this Q and K vectors we calculate attention and multiply it with V .So very roughly

attention = Softmax((**Q** * **K**) / Scale)
enbedding for attention layer = attention * **V**

This was just an overview on other blogs or tutorials for warmup. First these are not something special they are just Linear layers.

Simplified code for Q,K,V usage

As you see we are doing something very basic. We have linear layers and we are doing some operations on them. One more funny thing is input and output size of Linear layer is same. So we are not doing dimension reduction. What are we doing? Just learning!

When you begin to train a network, all weights are random, then by showing your training data, you make small updates according to your loss, with the

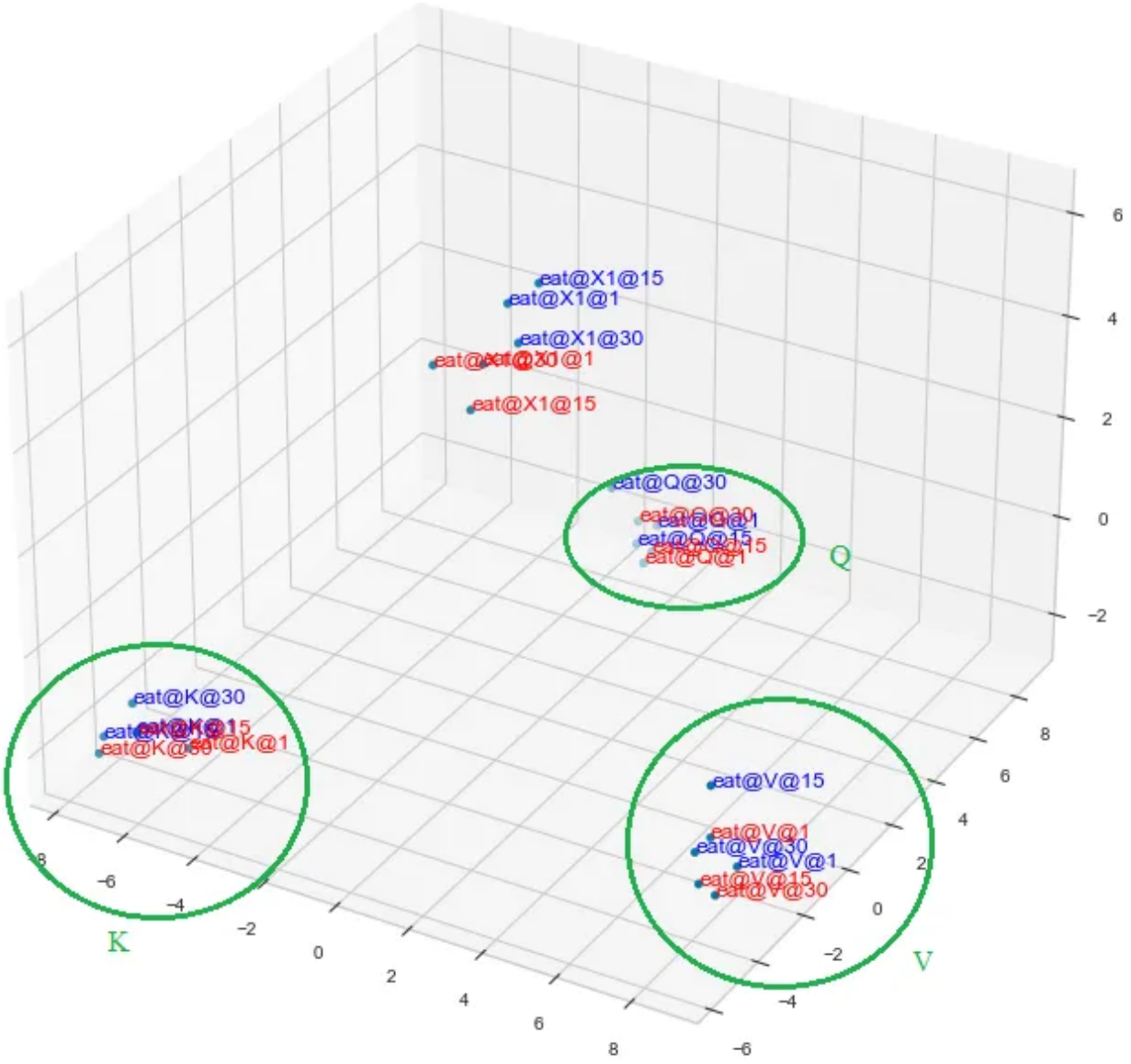
hope of having better weights after training.

So now I will show various visualizations of Q,K,V to understand what they do.

At encoder **Q,K,V** comes from encoder Embedding, so their embedding layer is for same language.(Src language English) Then they are vectors from same space. For 2 different sentences , I am collecting [Q,K,V,X1] vectors for time epochs [1,15,30] and dump on 3d or 2d.

```
energy = torch.matmul(Q, K.permute(0, 1, 3, 2)) / self.scale
...
attention = torch.softmax(energy, dim = -1)
...
x1 = torch.matmul(self.dropout(attention), V)
#x1 is the vector of attention multiplied by V, so it is what Q,K,V
learns
```

```
sentence1 = "we want to eat apple"
sentence2 = "we want to eat bread"
```



Q,K,V and x1 vectors traveling solution space for Encoder

Above image shows, the vectors from 2 sentences(different colors) and their clustering. Labels explanation :

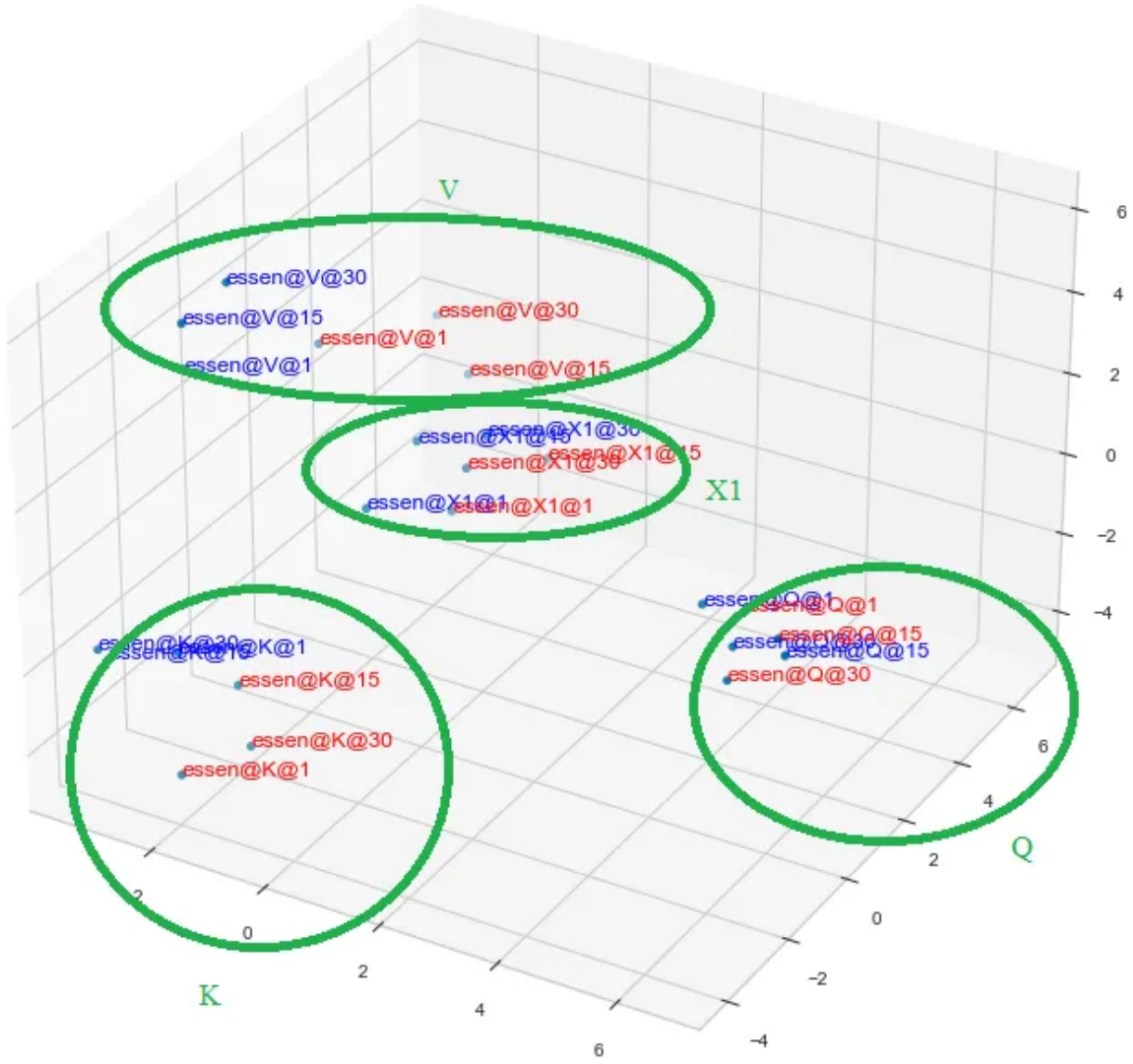
eat@k@30

eat : word of interest
K: vector
30: epoch

What does above chart shows? I have 2 sentences, and both have verb “eat”. In a language modeling words must have a meaning according to their context. So a good model must have different representation of **“eating bread”** and **“eating apple”**.

As you see Q,K,V forms their clusters and the result X1@30 for **“eat”** of sentence 1 becomes different than sentence 2. This was for encoder side.

We can do the same with decoder side.



Q,K,V and x1 vectors traveling solution space for Decoder

As you can see decoder side is more scattered. Because encoder has only 1 input type,(source language), but decoder side has his self-attention + encoder attention + position layer.

Can you understand the relation with Q,K,V and embedding of a word? Think that I have a network of only **Input -> Q -> K ->V -> Output**, and I feed my words into this network. At each epoch, I update weights of this network, according to loss(here multi class classification) network updates these Q,K,V(Linear layers) weights. Now at the end of training, the V output will be compliant with input word, right? That is how neural networks, learn, this is what happens here.

Let's sum now, in a big network architecture we have 3 Linear layers, Q,K,V and with these layers we create an embedding for our words. Why don't we learn only 1 encoding.

```
Think we have 3 values
10,20,30      -> Their multiplication is 6000
If we decrease every value
9 x 19 x 29   -> 4959
If we increase every value
11 x 21 x 21  -> 7161

As you see with little changes(with little updates to parameters by
backpropagation) we can make big differences.
```

Now let's try something,
Theory : Changing order of a sentence must yield same X4.
Why? Because Q,K,V vectors do not pay attention to position, and they are calculated by ,multiplying with all others. Think these as summarizing a book , or better expressing something in terms of features. A car is a

combination of wheels, windows, seats, brake. So creating Q,K,V is in a way feature extraction.

Also keep in mind that Q,K,V calculation is permutation invariant. That is by nature, I claim result vector X4 will be similar.(check code)

I claim that sentences **"we can eat apple"** and **"apple eat can we"** will have :
Q: Similar (at the end they will yield same sentence ,so at each step they will have similar Q),same entries at diagonal
K: Different, not all diagonals are different, but same items(apple, eat...) at different positions will have same embedding.(Reverse diagonal)
V: Different, not all diagonals are different, but same items(apple, eat...) at different positions will have same embedding.(Reverse diagonal)
X4: SAME , because at the end effect of multiplying(nearly multiplication with scale and softmax) 3 matrices will yield similar vector, because source sentence is same, and we generate output one by one.
Attention: Must be **same** for same items at different positions.

I generated Q,K,V vector of 2 sentences and get cosine_similarity of all combinations. Check diagonals to understand how same embedding created for same items at different positions. Columns and indexes are same because both sentences created the same output.

Decoder Q

	sos->wir	wir->konnen	konnen->apfel	apfel->essen	essen->eos
sos->wir	1.00	0.25	-0.09	-0.14	-0.33
wir->konnen	0.25	1.00	-0.30	-0.09	-0.27
konnen->apfel	-0.09	-0.30	1.00	-0.16	-0.18
apfel->essen	-0.14	-0.09	-0.16	1.00	0.19
essen->eos	-0.33	-0.27	-0.18	0.19	1.00

Vector : decoder_encoder_attention@K

	sos	apple	eat	can	we	eos
sos	1.00	0.46	0.29	0.10	-0.26	0.65
we	-0.30	-0.27	0.09	-0.29	0.99	-0.19
can	0.09	-0.03	0.09	1.00	-0.27	-0.05
eat	0.32	0.45	1.00	0.06	0.08	0.33
apple	0.48	0.99	0.41	-0.02	-0.25	0.65
eos	0.65	0.62	0.30	-0.04	-0.16	1.00

decoder encoder_attention@V

	sos	apple	eat	can	we	eos
sos	1.00	0.24	0.16	0.16	-0.23	0.50
we	-0.26	-0.14	-0.07	-0.16	0.99	-0.36
can	0.15	0.01	0.15	1.00	-0.12	0.25
eat	0.18	0.19	1.00	0.12	-0.10	0.21
apple	0.26	1.00	0.16	0.04	-0.13	0.63
eos	0.50	0.60	0.20	0.26	-0.34	1.00

Q,K,V of Decoder for 2 sentences.

Decoder X4

	sos->wir	wir->konnen	konnen->apfel	apfel->essen	essen->eos
sos->wir	0.99	-0.17	-0.32	0.05	-0.39
wir->konnen	-0.13	1.00	0.25	0.23	0.27
konnen->apfel	-0.28	0.28	1.00	0.47	0.97
apfel->essen	0.04	0.22	0.48	1.00	0.45
essen->eos	-0.36	0.29	0.96	0.44	1.00

Decoder Attentions

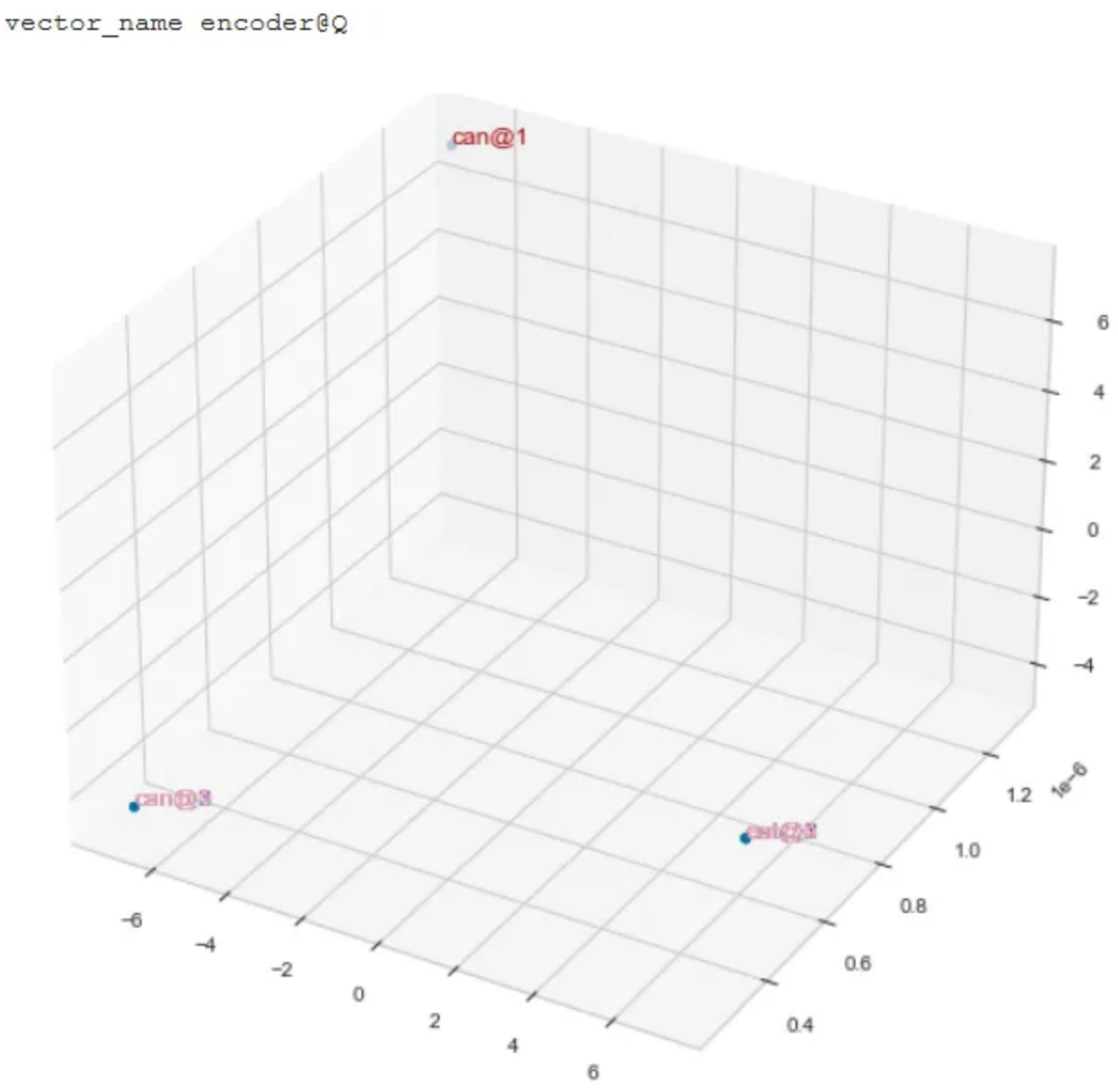
	sos->wir	wir->konnen	konnen->apfel	apfel->essen	essen->eos
sos->wir	0.01	0.02	0.68	0.07	0.09
wir->konnen	0.04	0.09	0.07	0.98	0.06
konnen->apfel	0.67	0.08	0.62	0.28	0.72
apfel->essen	0.08	0.98	0.25	0.11	0.18
essen->eos	0.10	0.06	0.71	0.19	1.00

Decoder x4 and attentions

By checking above tables, you can understand relation of these vectors. Also you can try for different sentences, different combinations.

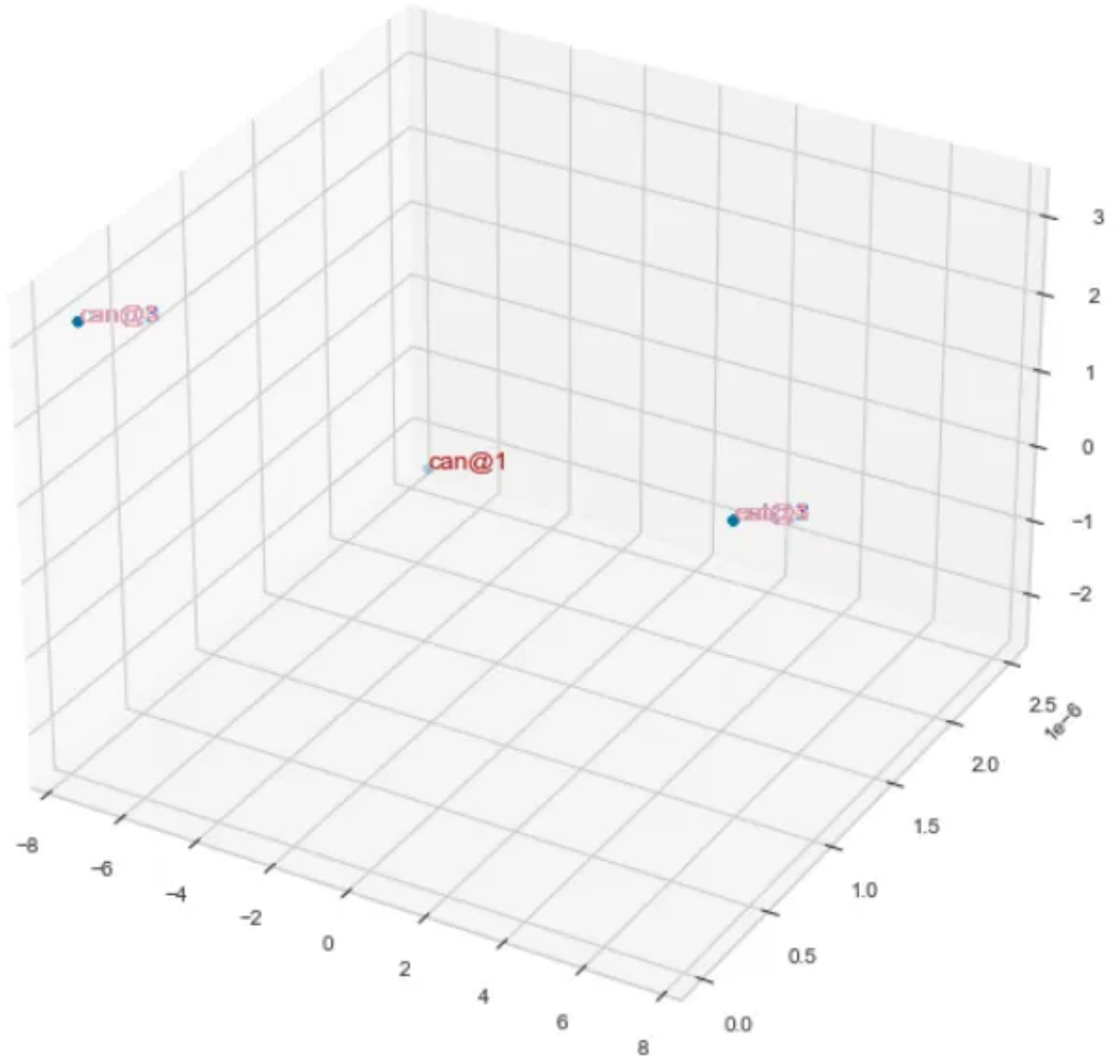
Now let's do another sample. For 5 sentences I will get Q,K,V,X4 vectors, and check how they are positioned in space. Sentence 1 and 2 are proper, 3 and 4 are nonsense. Last sentence both have "apple" and "book" so it is there to mislead the network.

```
#The translations of 5 sentences
source = ['i', 'can', 'eat', 'apple']
predicted target = ['ich', 'konnen', 'apfel', 'essen', '<eos>']
source = ['i', 'can', 'eat', 'bread']
predicted target = ['ich', 'konnen', 'brot', 'essen', '<eos>']
source = ['i', 'can', 'eat', 'book']
predicted target = ['ich', 'konnen', 'apfel', 'essen', '<eos>']
source = ['i', 'can', 'eat', 'newspaper']
predicted target = ['ich', 'konnen', 'brot', 'essen', '<eos>']
source = ['i', 'can', 'eat', 'apple', 'book']
predicted target = ['ich', 'konnen', 'apfel', 'essen', '<eos>']
```



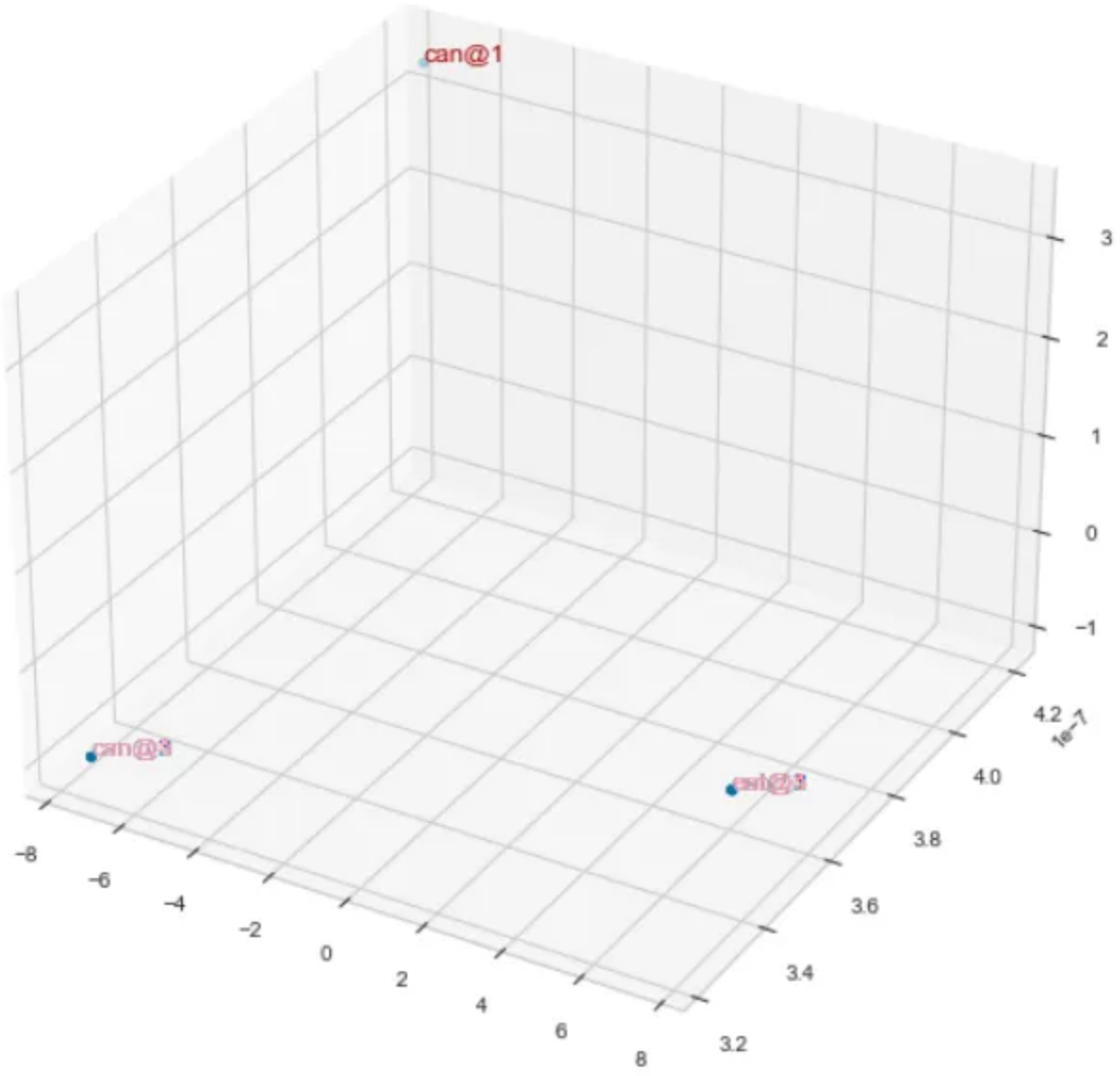
Q vector

```
vector_name encoder@K
```

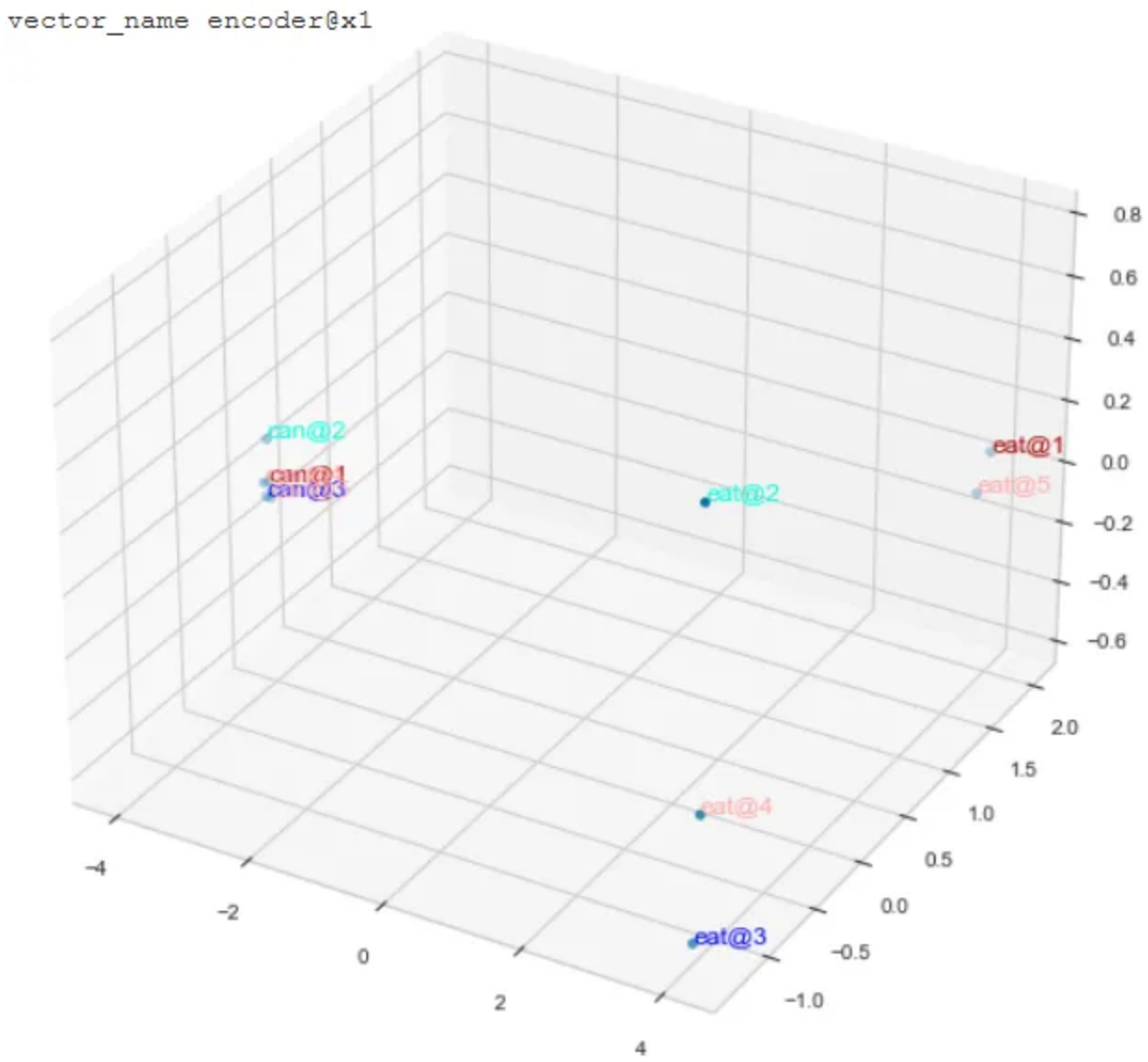


K vector

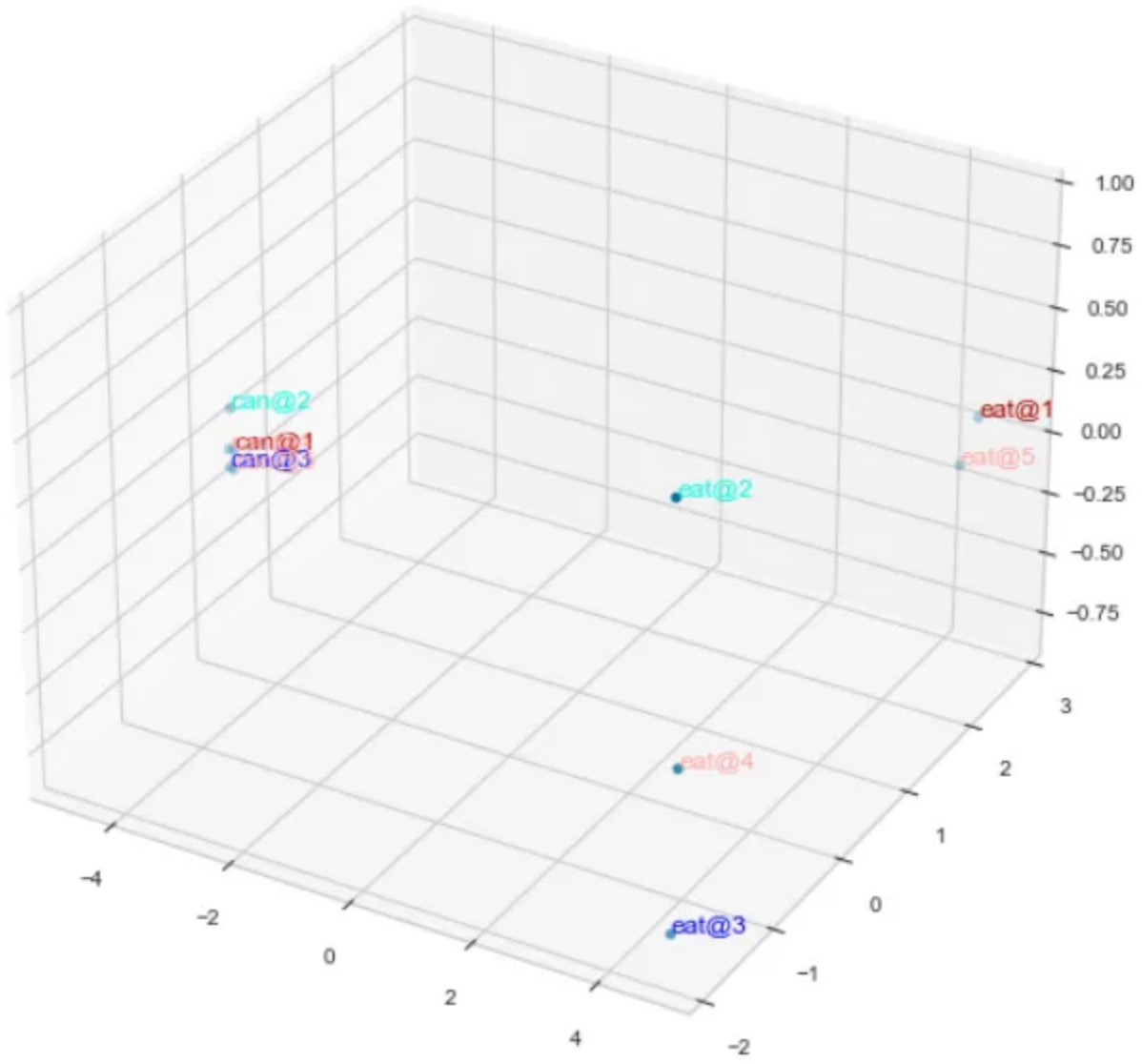
```
vector_name encoder@V
```

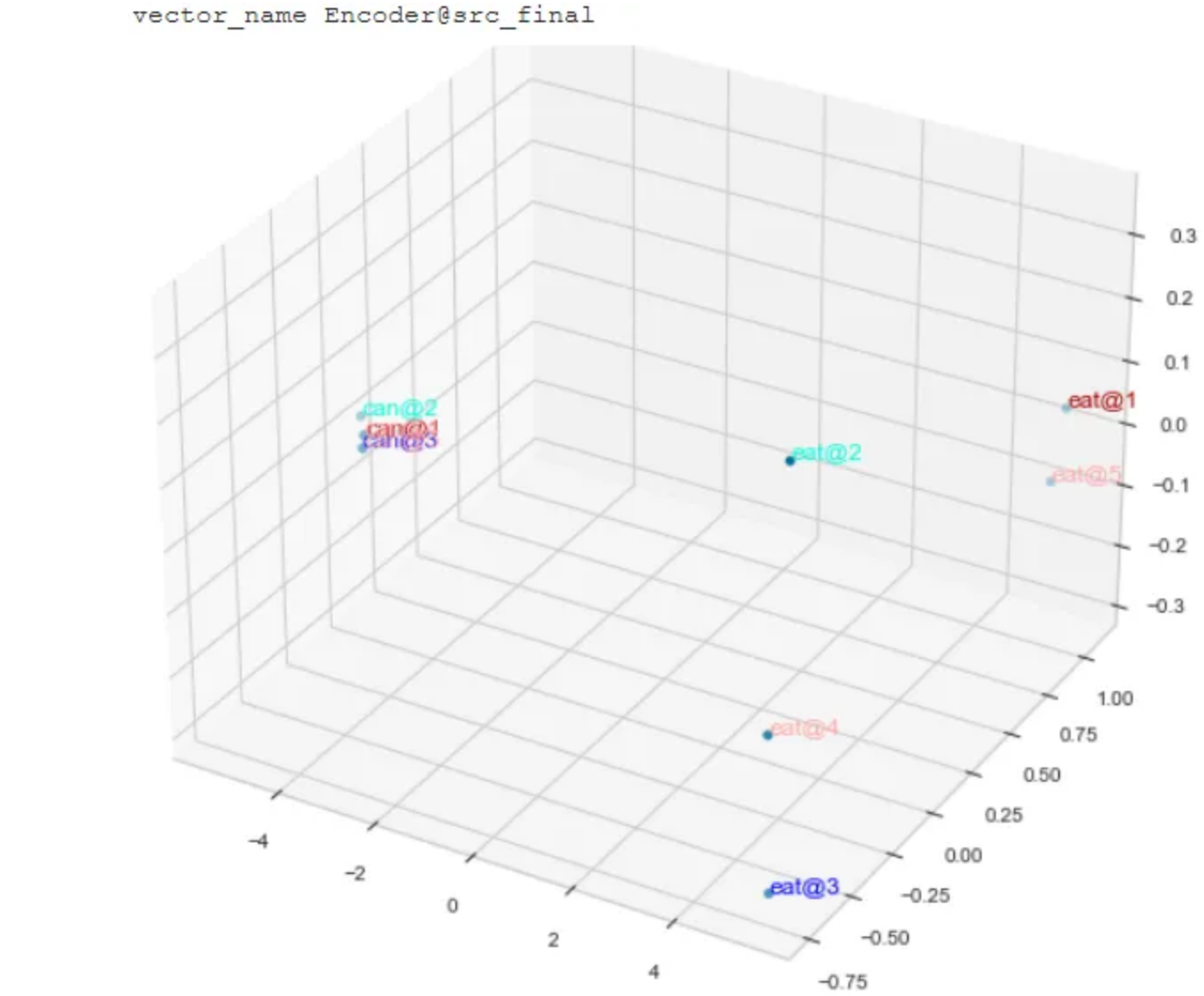


V vector

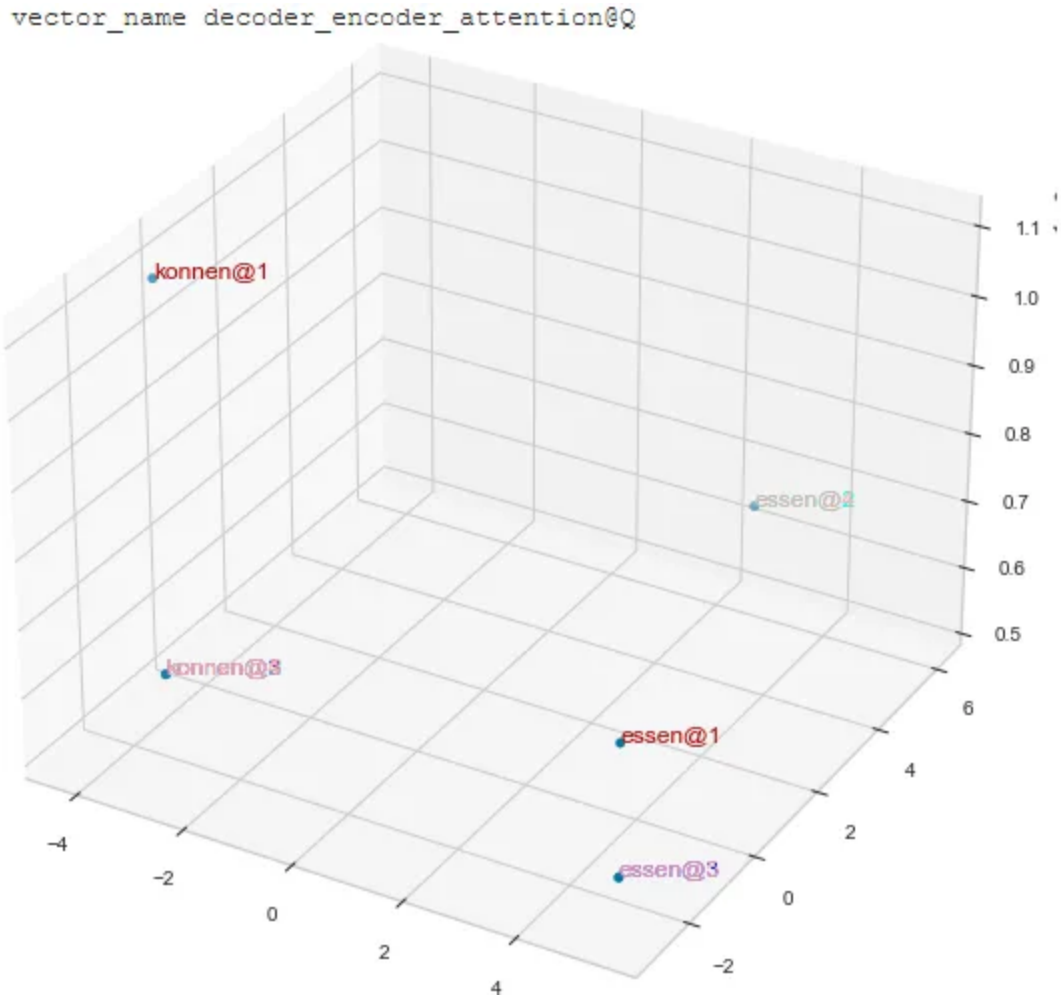

vector_name encoder@x1
can@2
can@1
can@3
eat@2
eat@1
eat@5
eat@4
eat@3
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−4
−2
0
2
4

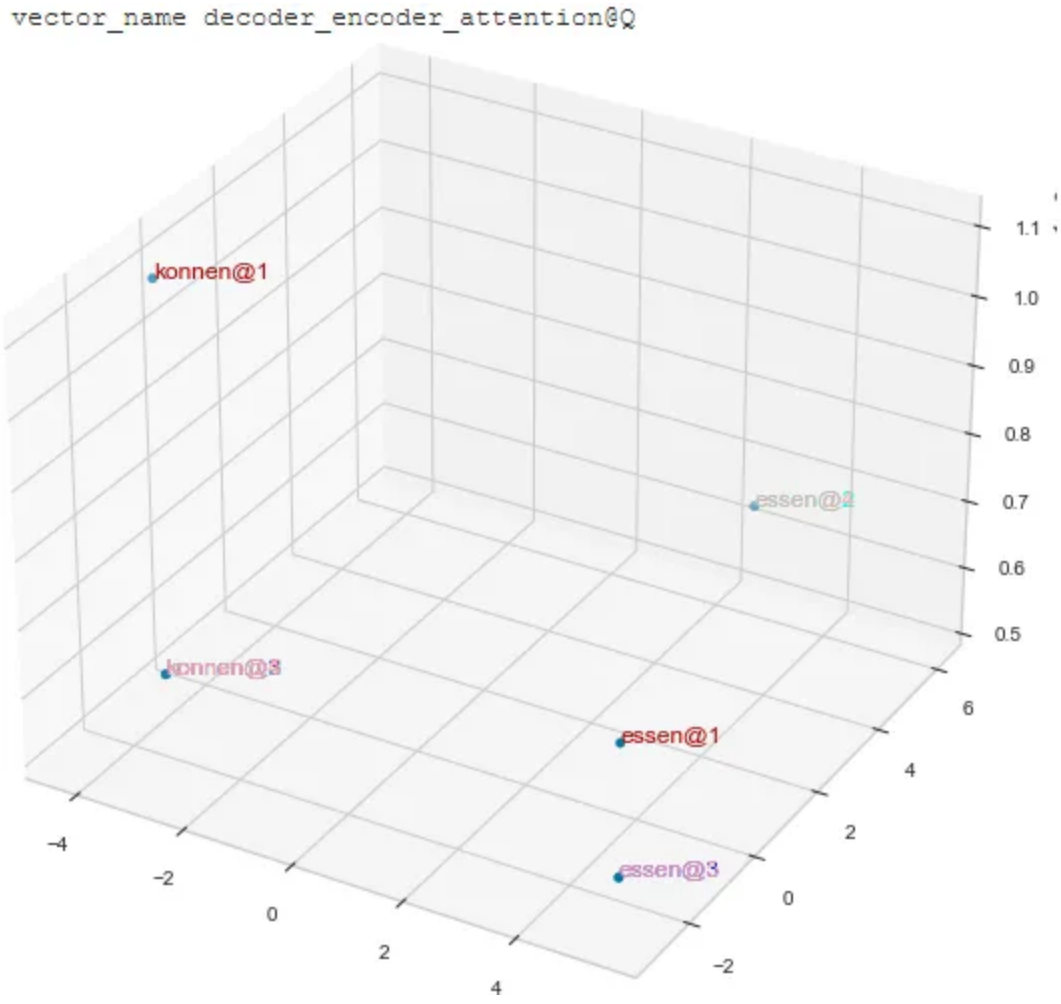
```
vector_name encoder@x4
```

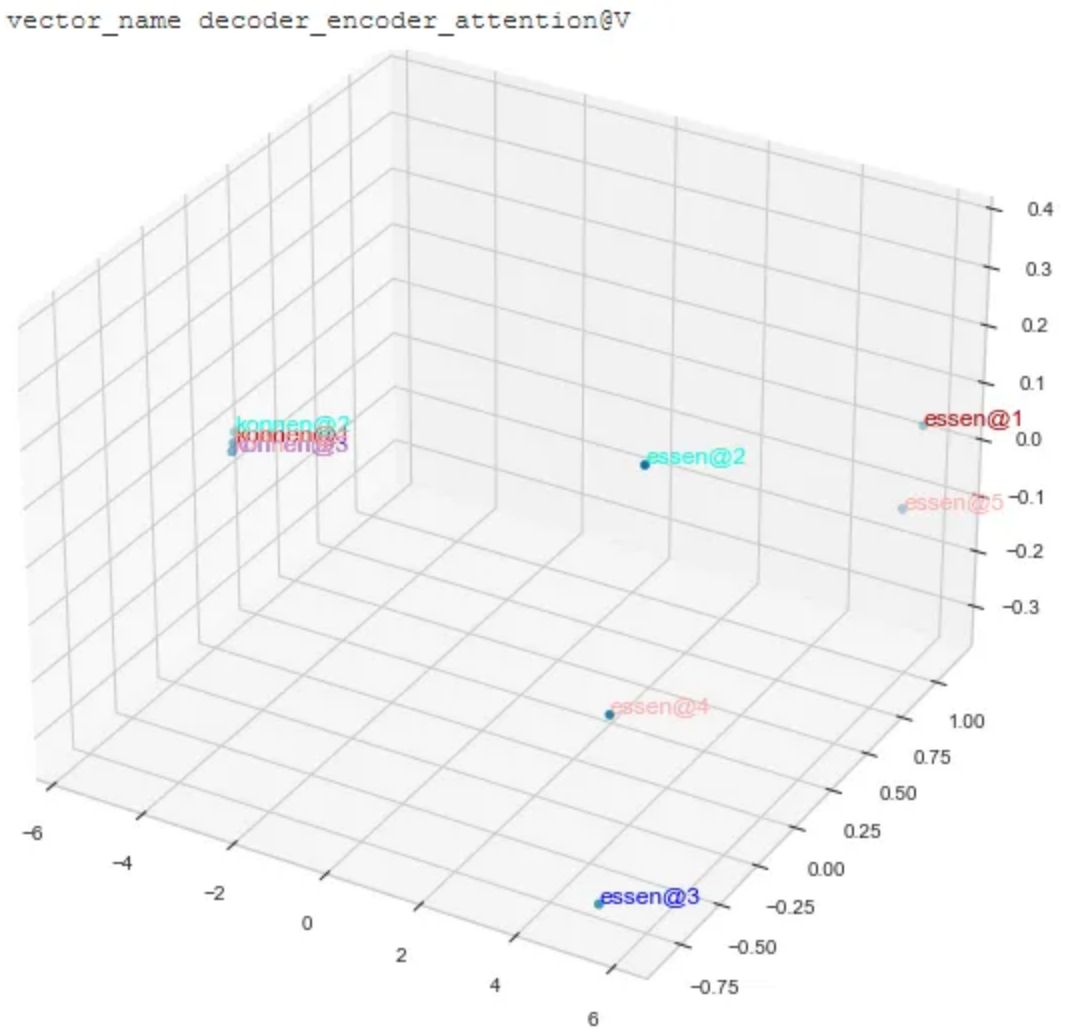




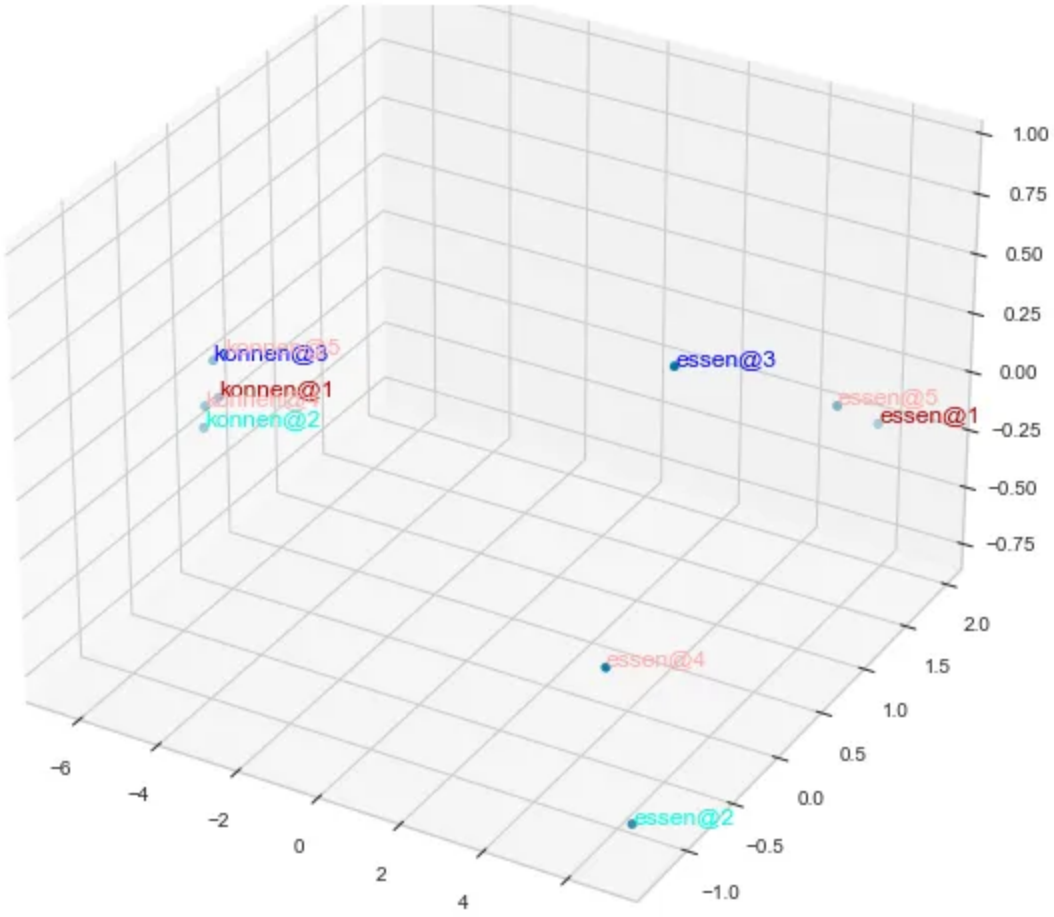
As you can see , **Q,K,V** vectors seem so similar but results(**X1,X4,src_final**) are scattered. Because as I said maybe 1 vector is not so different from others in it's category but their combined effect is different. We can also show it for decoder part.

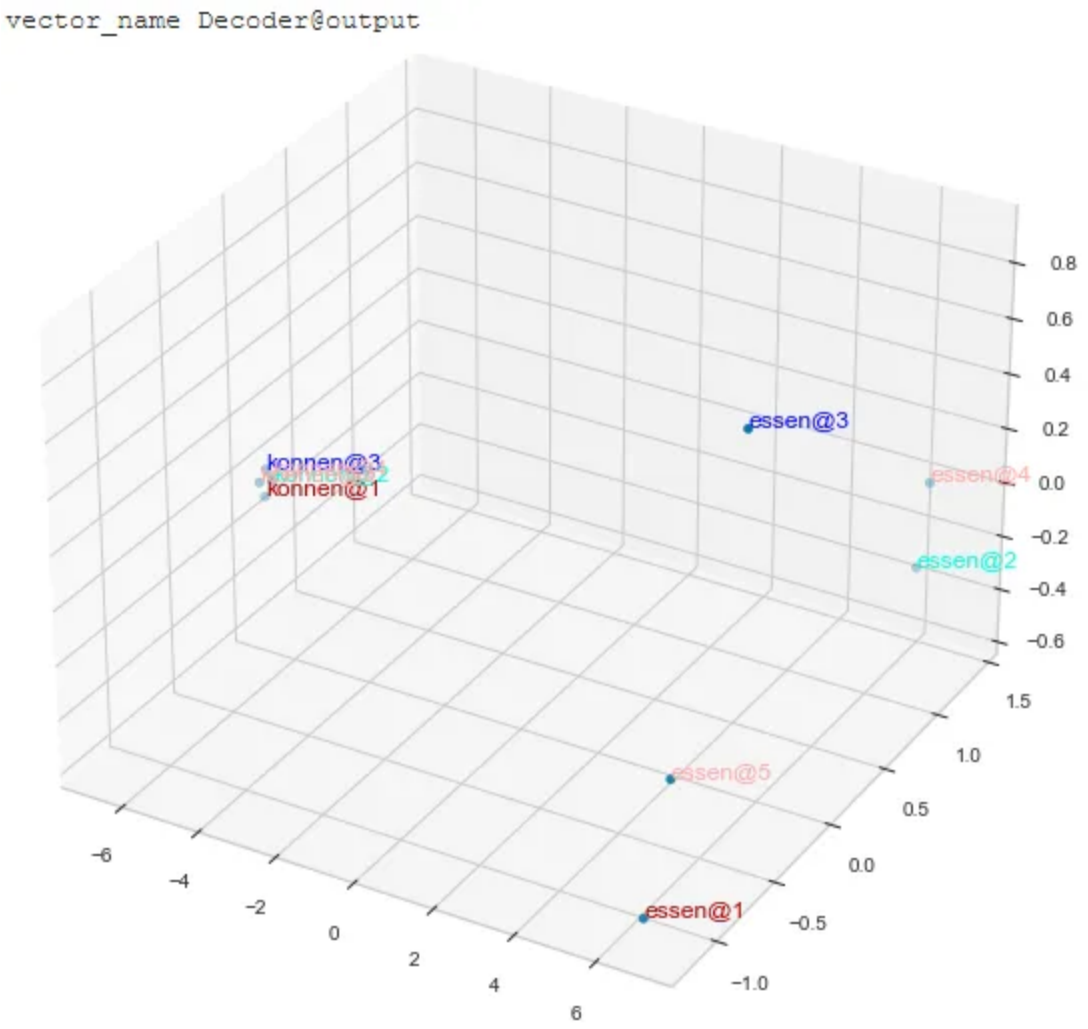

vector_name decoder_encoder_attention@Q
konnen@1
essen@2
konnen@3
essen@1
essen@3
1.1
1.0
0.9
0.8
0.7
0.6
0.5
−4
−2
0
2
4
6
4
2
0
−2


vector_name decoder_encoder_attention@Q
konnen@1
konnen@3
essen@2
essen@1
essen@3
1.1
1.0
0.9
0.8
0.7
0.6
0.5
−4
−2
0
2
4
6
4
2
0
−2


vector_name decoder_encoder_attention@V
essen@1
essen@2
essen@3
essen@4
essen@5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−6
−4
−2
0
2
4
6

```
vector_name decoder_encoder_attention@x4
```





As you see , decoder has more scattered vectors, because it has more source of data,(including more and diverse vectors in calculation)

With samples above I showed, what we are doing with these vectors is creating good representations of words. How do we know “water” is “drinkable” but “apple” is edible ? Because we see lots of sentence for these as pairs. So when you see a self-attention logic, u will understand that by time, network learns which co-occurrences are by random which are not. And then create good vectors of the input according which depends on context.

Machine Learning NLP Self Attention Transformers

More from the list: "NLP"

Curated by Himanshu Birla

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

· 11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

· 6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

· 15 min rea

View list

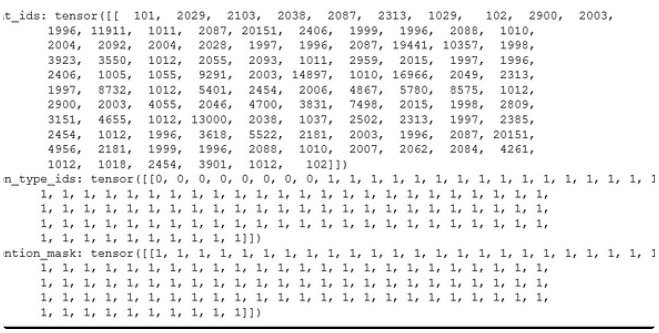
Written by mustafac

65 Followers · Writer for Analytics Vidhya

Data Scientist & Machine Learning

Follow

More from mustafac and Analytics Vidhya



mustafac in Analytics Vidhya

Bert For Question Answering

In this post I will show the basic usage of “Bert Question Answering” (Bert QA) and in the...

6 min read · May 7, 2021

6

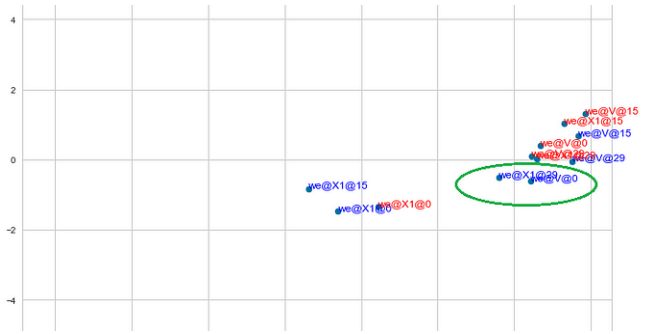
Sajal Rastogi in Analytics Vidhya

Wifi -Hacking using PyWifi

.

4 min read · Feb 6, 2021

252 1



Kia Eisinga in Analytics Vidhya

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open...

7 min read · Jan 27, 2020

2K 24

mustafac in Analytics Vidhya

NLP Transformer Unit Test

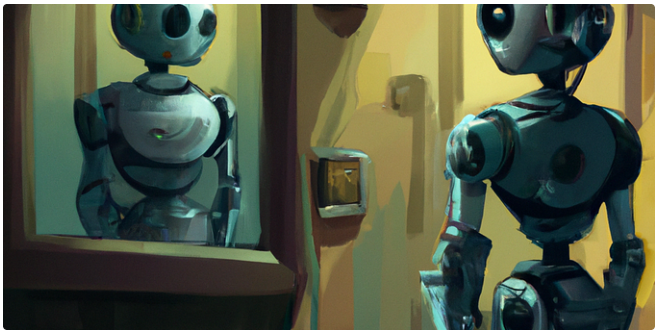
In Machine Learning, it is hard to visualize or test something small. When it is NLP, domai...

11 min read · Jan 1, 2021

7

See all from mustafac See all from Analytics Vidhya

Recommended from Medium



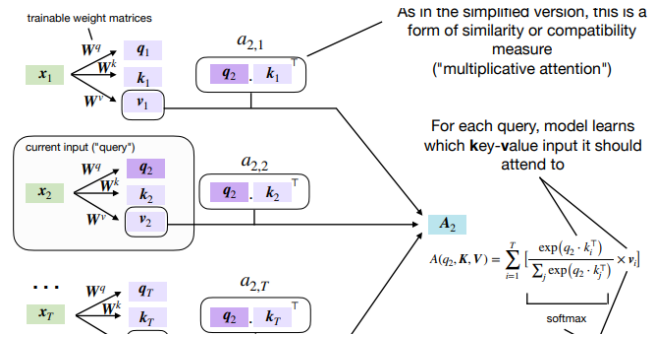
Thomas van Dongen in Towards Data Science

Demystifying efficient self-attention

A practical overview

20 min read · Nov 7, 2022

477 2



Zain ul Abideen

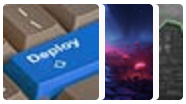
Attention Is All You Need: The Core Idea of the Transformer

An overview of the Transformer model and its key components.

6 min read · Jun 26

144

Lists



Predictive Modeling w/ Python

20 stories · 452 saves

Natural Language Processing

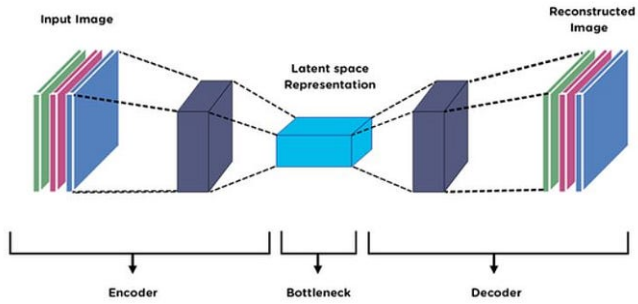
669 stories · 283 saves

Practical Guides to Machine Learning

10 stories · 519 saves

The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves

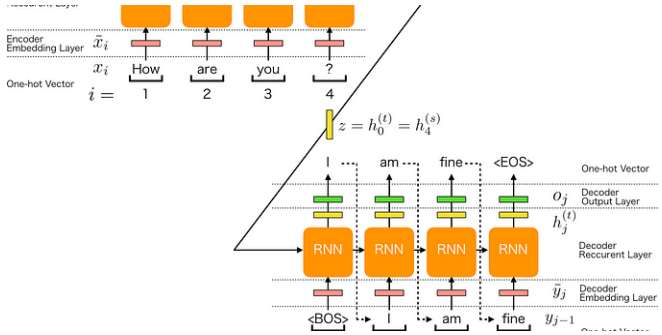


Ahmadsabry

A Perfect guide to Understand Encoder Decoders in Depth with...

Introduction

6 min read · Jun 24



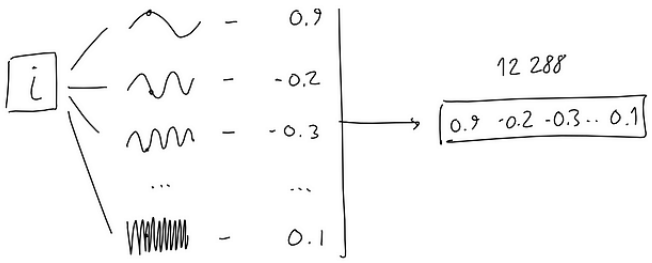
Eugene Ku

Transformer Architecture (Part 2—Self-Attention)

Last time, we learned how Transformers utilize Positional Encoding to obtain the abili...

7 min read · Aug 23

10



Rand in AI Mind

Creating Sinusoidal Positional Embedding from Scratch in...

Recent days, I have set out on a journey to build a GPT model from scratch in PyTorch....

6 min read · Jun 28

103

Michael Humor in CoinsBench

What are the LLaMA model weights?

The LLaMA models released by Meta AI were trained with different transformer...

4 min read · May 3

3 1

See more recommendations