## horizontal line

Multi Head Attention , Positional Encoding and Layer Normalization

10.06.2025

# Multi Head Attention

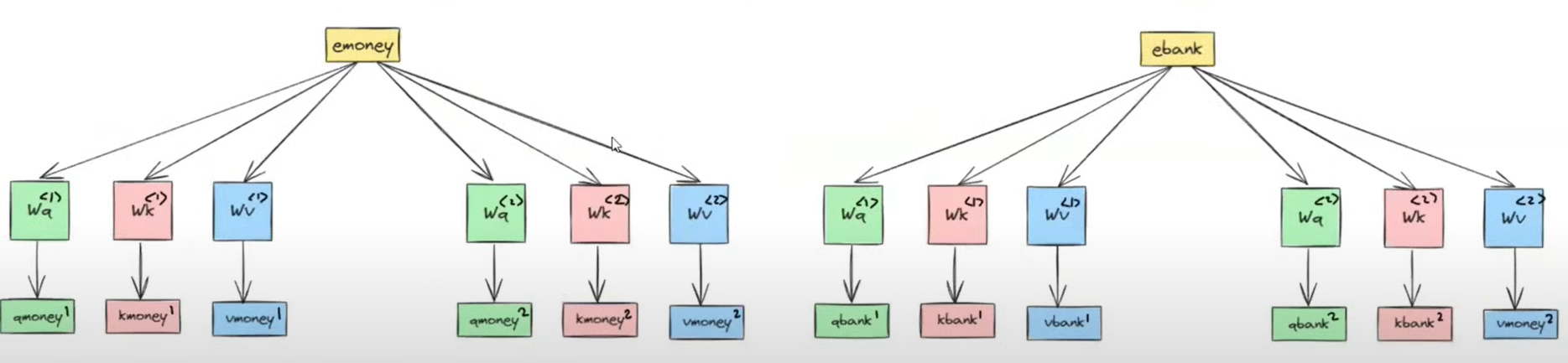
A sentence may have different contexts that cannot be captured by self attention that only capture a single context. A self attention is a head and we use multiple heads to capture different meanings.

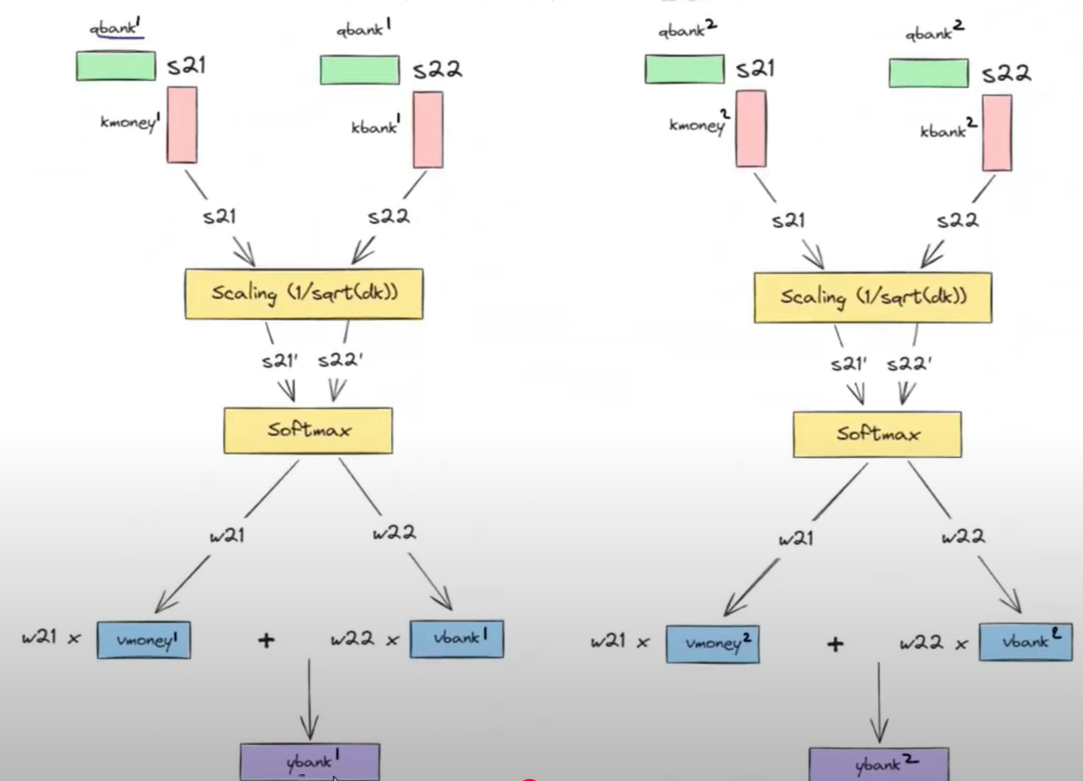
E.g. The man saw the astronomer with a telescope.

Based on the similarity between “man” , “astronomer” and “telescope” we get different meanings.

# How do we apply it ??

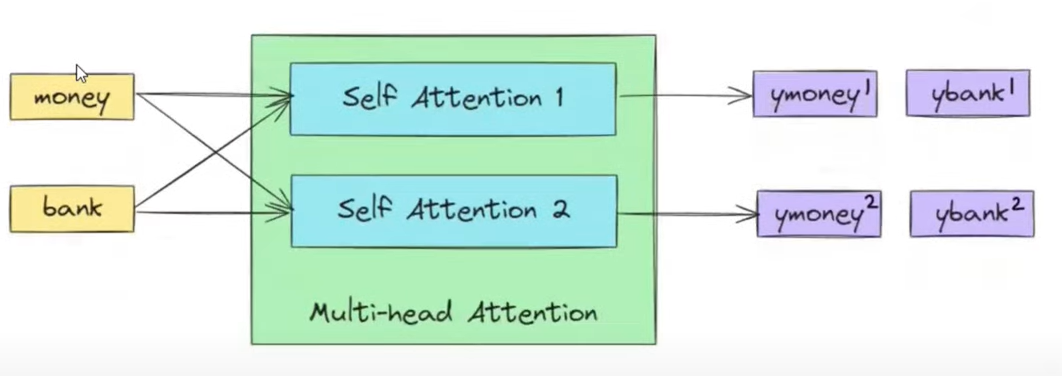
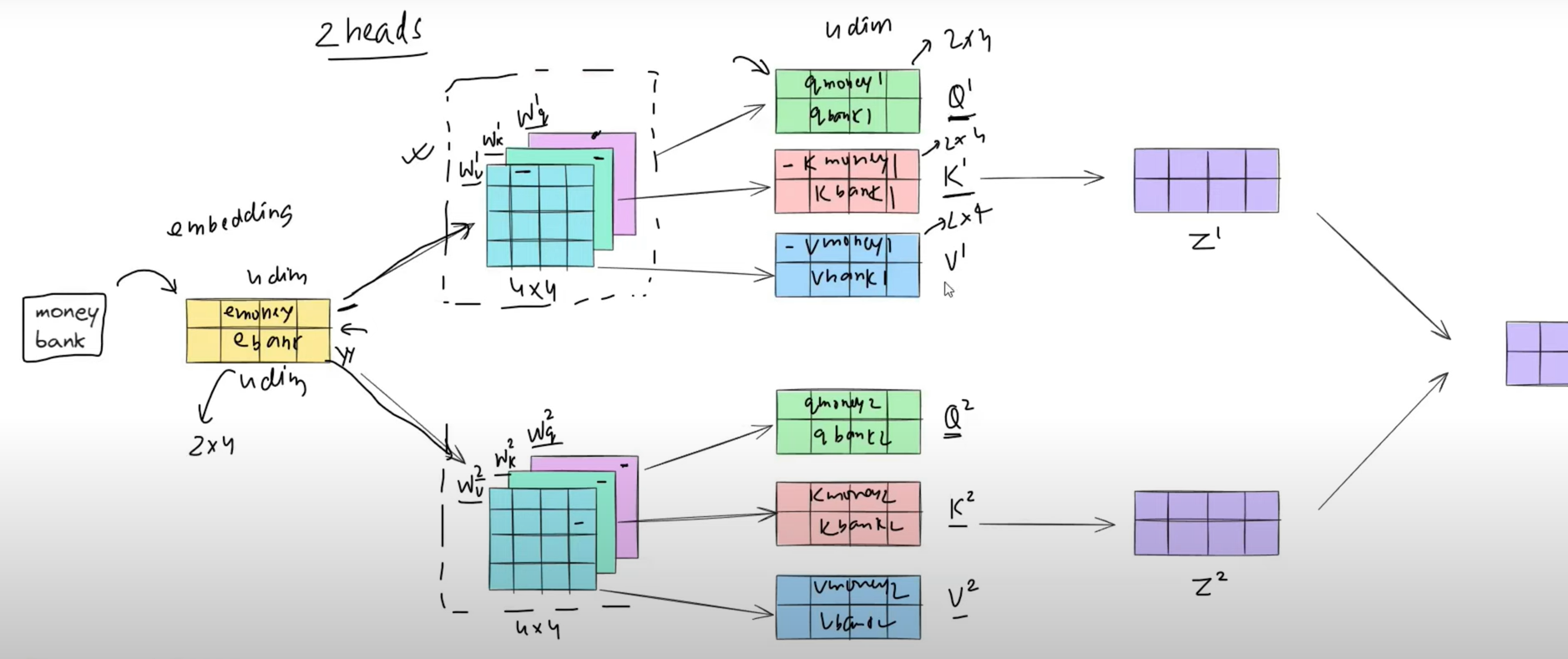
We can do it by introducing one more set of matrix “” , “” , “” and “” , “”, “”. Thus from it we get 2 different sets of vectors.

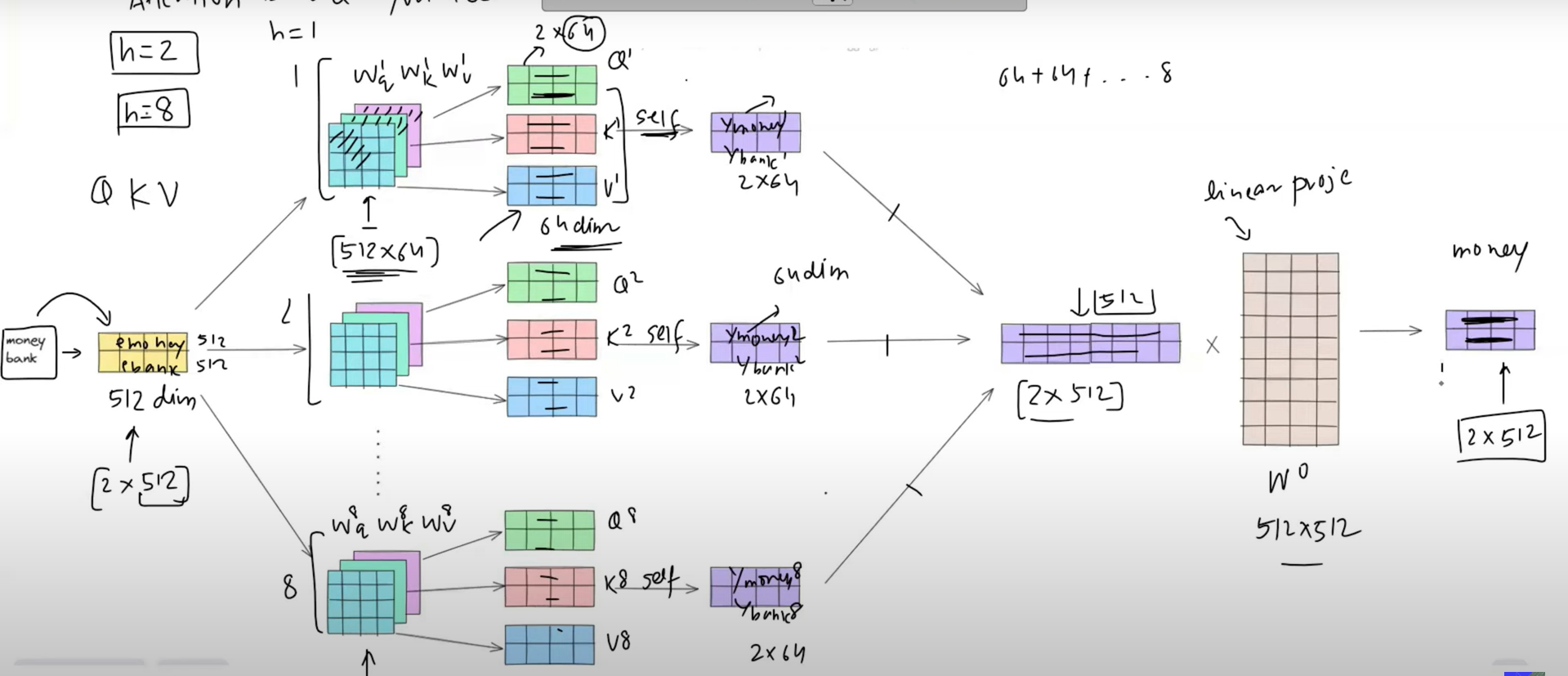


We get two contextual embeddings that combine by linear transformation where . We have random weights in the matrix that are trained during training.

In transformers we use 8 heads .

We have 512 dim embeddings that are divided between 8 heads reducing computation and each head gets 1/8 th of dimension.





# Positional Encoding

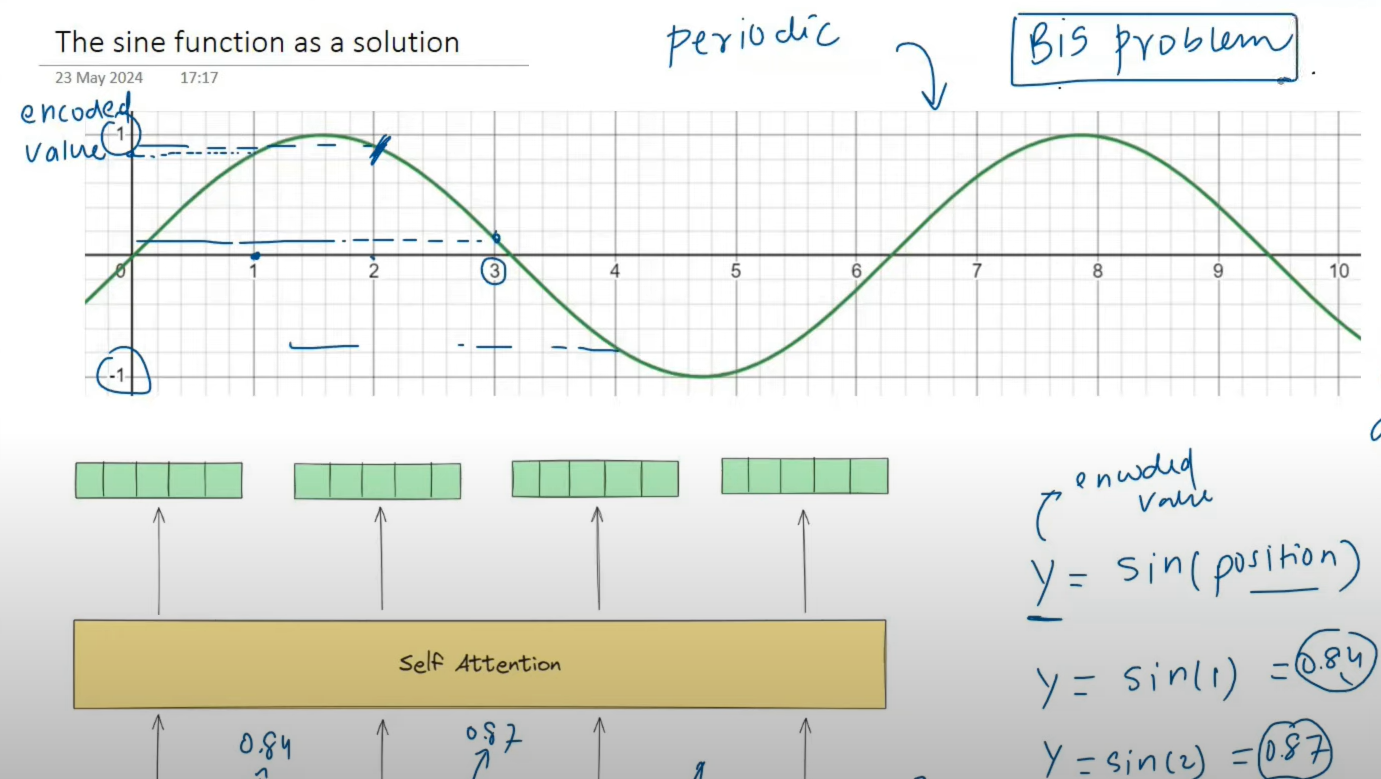
To add position vector to the word embedding because sequence matters and attention lacks in it.

One simple solution is to blindly add pos at the end of the vector but that will be an unbounded range towards the upper limit causing problems in backpropagation.

Problems :

1. Unbounded
2. Discrete values
3. Can’t capture relative position.

Sin and Cos functions can replace them.

What we can do is to find 

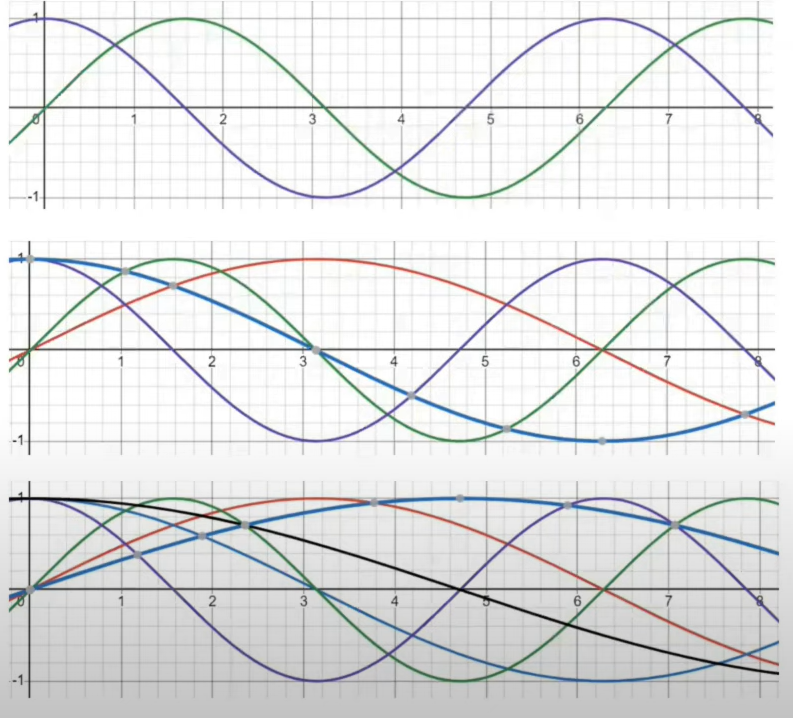
Y = sin(position) that can be added.

But the problem is that the position of each vector should be unique to translate correctly but sin is a periodic function that repeats.

A solution was introduced to also use the Cos function and make a 2-D vector of it. 

[Sin(pos) , Cos(pos)]

Reduces the chances of repetition .

For more, make new pairs of Sin and Cos.

2-D , 4-D , 6-D as the same dimension of the word embedding.

E.g. River(6\_dim) Bank(6\_dim)

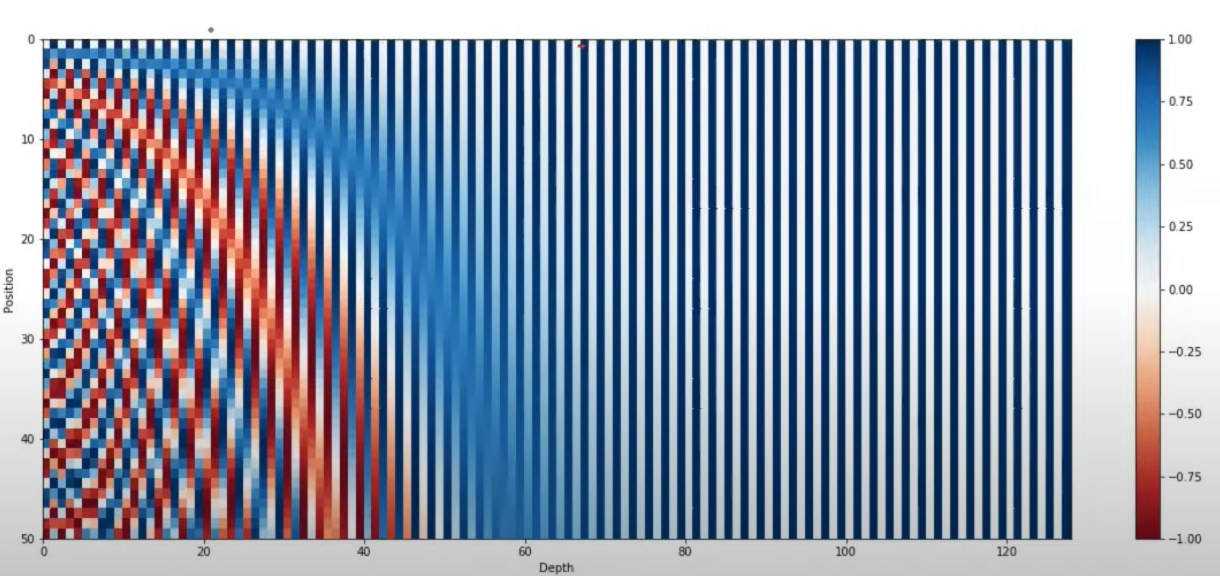
+

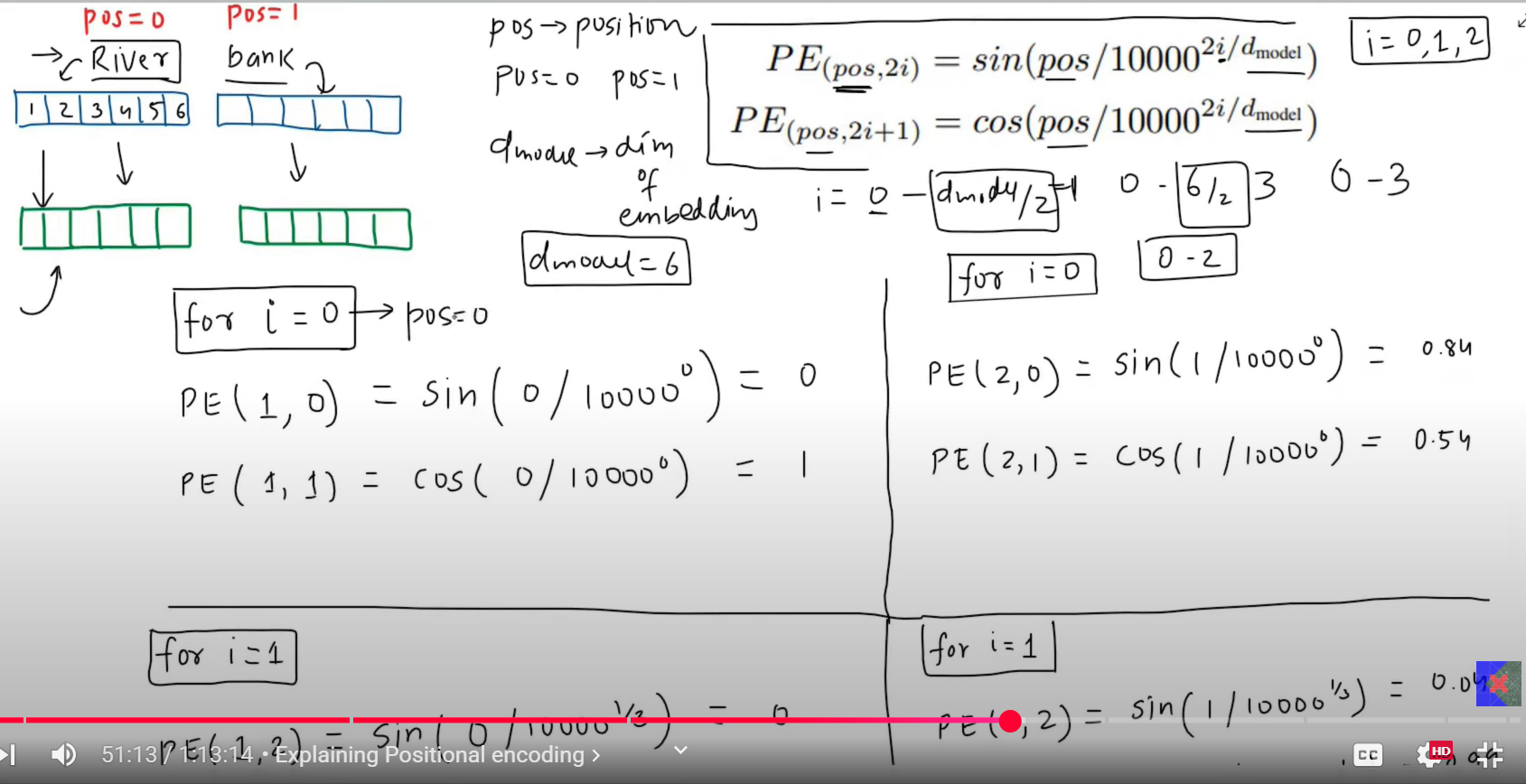
Pos Enc(6-dim) pos Enc(6-dim)

Resultant also a 6\_dim vector. We do not concatenate them because that would increase computational complexity by 12-dim.

[ \_ , \_ , \_ , \_ , \_ , \_ ] is the pos Encoding. Starting 2 will be Sin(pos) and Cos(pos)

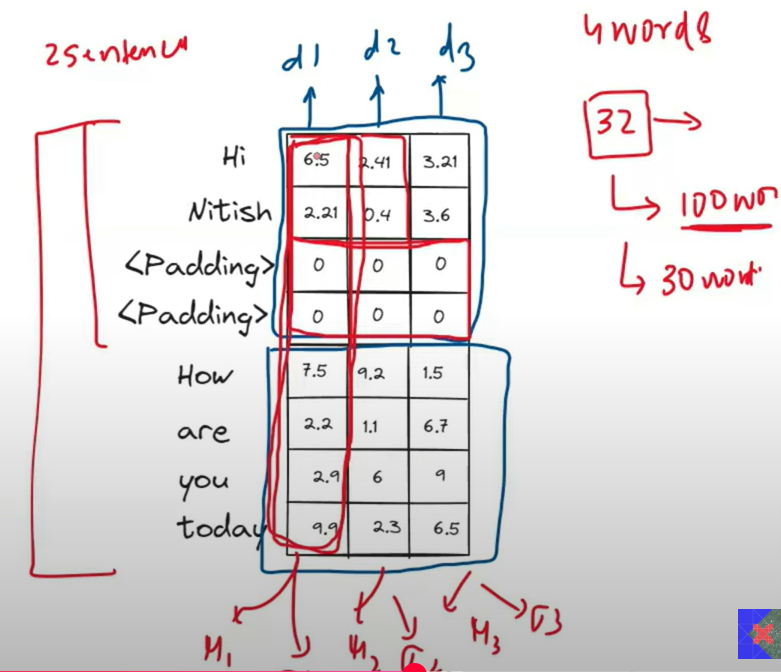
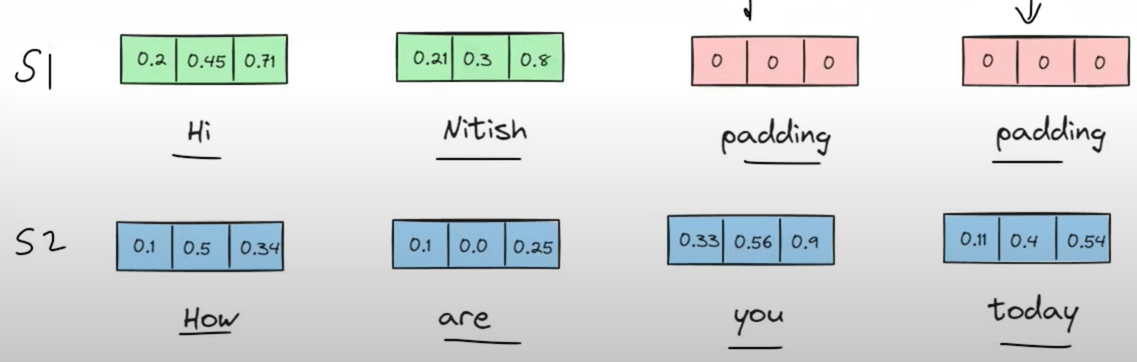
Next2 will be Sin(pos/2) and Cos(pos/2) then next2 , Sin(pos/3) and Cos(pos/3). This is how it goes on Reducing chances to repeat very negligibly.



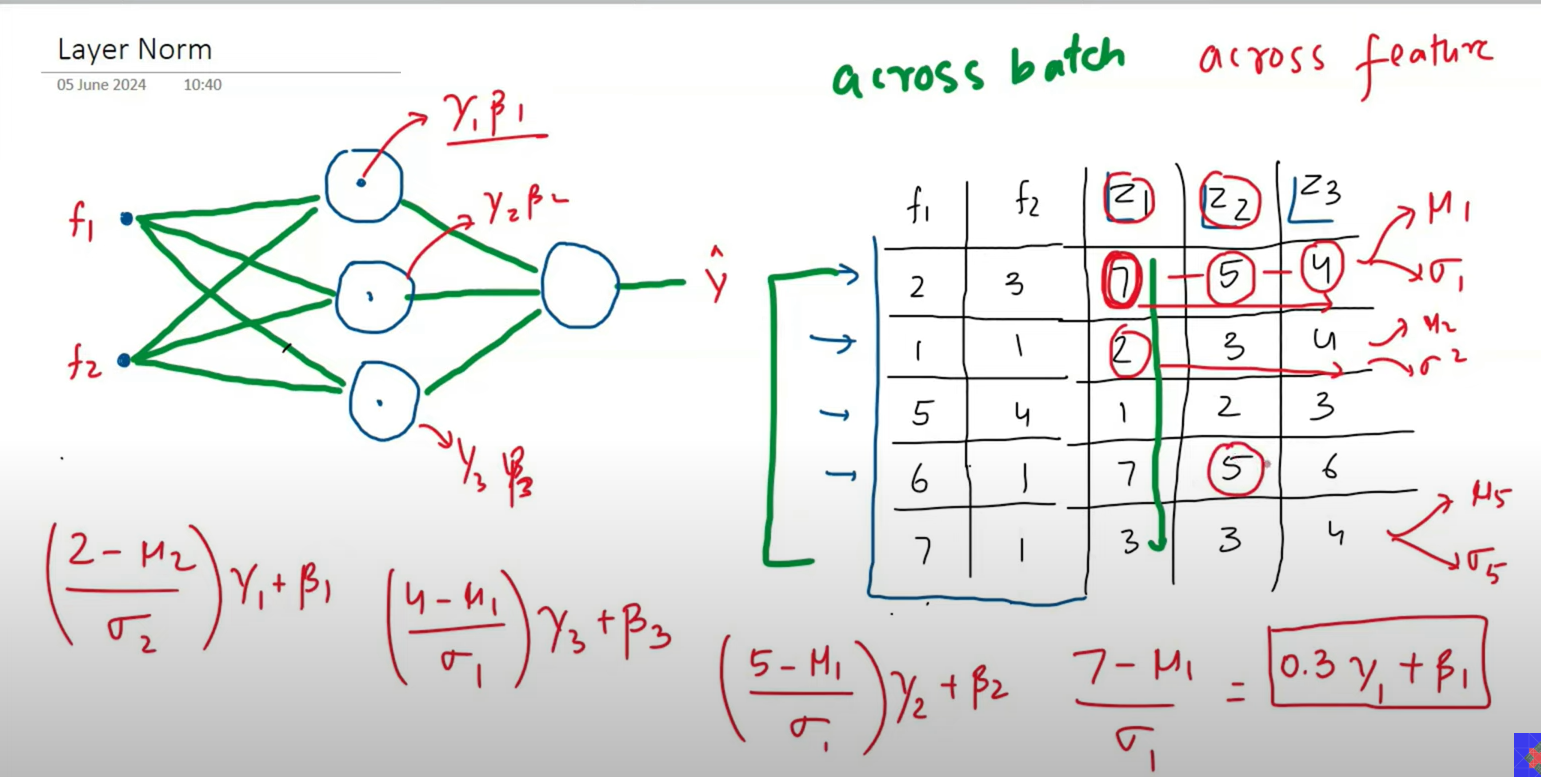
As we go for high dimensions, frequency reduces.

# Layer Normalization

We do it in transformers because it is done across features while avoiding Batch Normalization because that is done across Batches.

The main problem in using Batch Normalization in Transformers is , If we have 32 sentences in a batch with max words 100 and min words 30 so every time a lot of padding will be added. 

Thus to calculate the mean of such data having a lot of zeros would give an incorrect mean.



Thus layer normalization is done because it calculates mean across features.

This would cause an impact on padded rows since it is calculated individually.

