Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

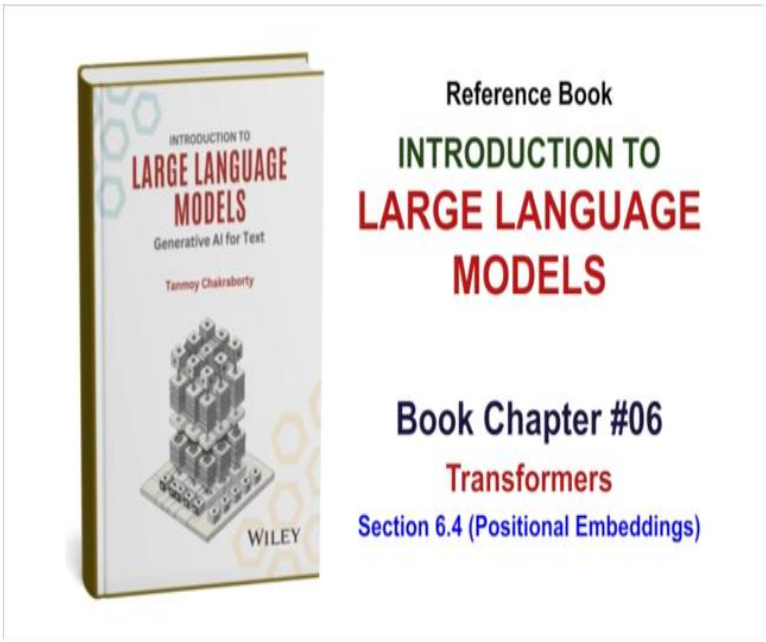
Department of Computer Science & Engineering

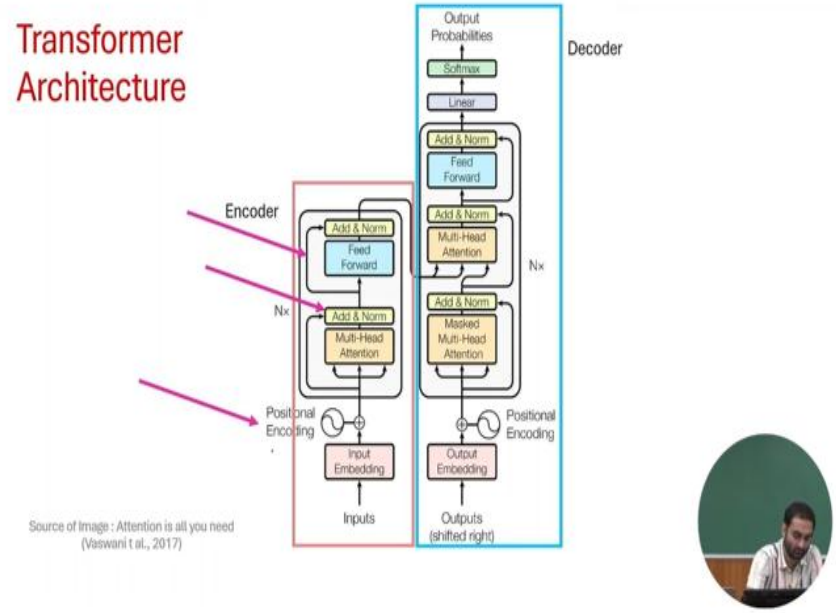
Indian Institute of Technology, Delhi

Lecture 16

Introduction to Transformer: Positional Encoding and Layer Normalization

In the last class we started discussing different components of the transformer model and so far we discussed attention self-attention we noticed that the self-attention operation is a linear operation therefore we had to include some sort of non-linearity then we added feed forward network right feed forward network was a position wise feed forward network right so Then, you know, in the encoder block, we have, I mean, in each of the encoder blocks, we'll have self-attention layer and the feedforward layer, right? In the decoder part of it, in every block, there's a self-attention, masked self-attention, right? Because the decoder block should not be aware of what is going to come in the future. In the normal self-attention every attention head has access to all the other positions right but in mask self-attention the attention head has access to only the left side of the representations not the right side okay and we also discussed that you know in order to basically get an interaction between the encoder and decoder layers we need a cross attention right in which the key will come the query will come from the decoder part and key and value will come from the encoder part right and then to that through a normal self-attention mechanism, we will let the decoder interact with the encoder. That's what we discussed. We also mentioned in the last class that transformers in transformer model accesses all the tokens in parallel right the position information is not properly maintained right so in order to you know let the model understand the position of every token we need an additional tokens which is called position embedding right the additional token embedding called positional embedding so we'll discuss positional embedding in today's class





we will first discuss the default positional embedding method proposed by Vaspani et al in the transformer paper which is the sinusoidal positional embedding and then we move to the recent one which is the rotary positional encoding probe. And then we will also talk about some other small components in transformer like add and norm layer, residual

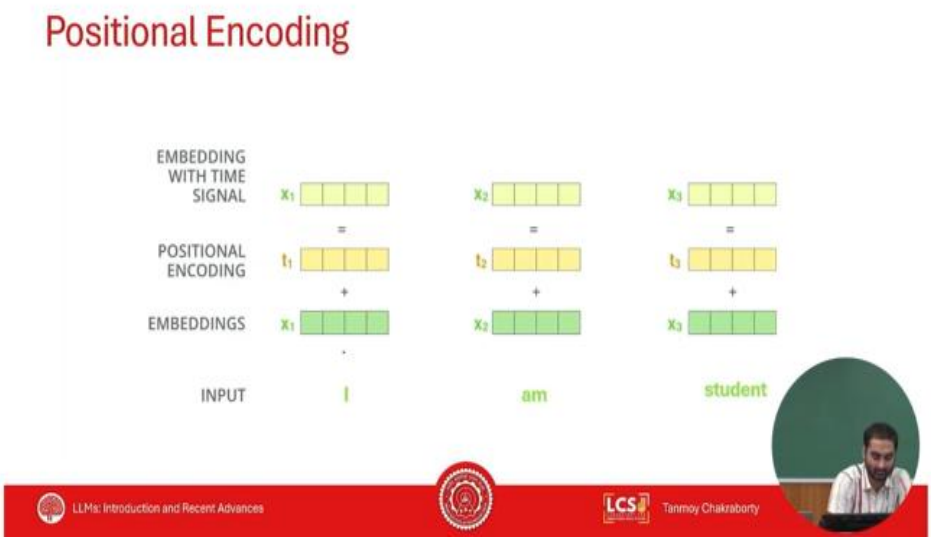
connection and add and norm layer. In the transformer model, today we will focus on positional encoding, add norm layer and this residual connection.

Position Information in Transformers: An Overview

Philipp Dufter, Martin Schmitt, Hinrich Schütze

Abstract

Transformers are arguably the main workhorse in recent natural language processing research. By definition, a Transformer is invariant with respect to reordering of the input. However, language is inherently sequential and word order is essential to the semantics and syntax of an utterance. In this article, we provide an overview and theoretical comparison of existing methods to incorporate position information into Transformer models. The objectives of this survey are to (1) showcase that position information in Transformer is a vibrant and extensive research area; (2) enable the reader to compare existing methods by providing a unified notation and systematization of different approaches along important model dimensions; (3) indicate what characteristics of an application should be taken into account when selecting a position encoding; and (4) provide stimuli for future research.



So, let us start with the positional encoding layer. So, this is the survey paper that I would strongly recommend you guys to read. This contains more or less all sorts of positional encoding methods proposed so far. The archive version is more updated.

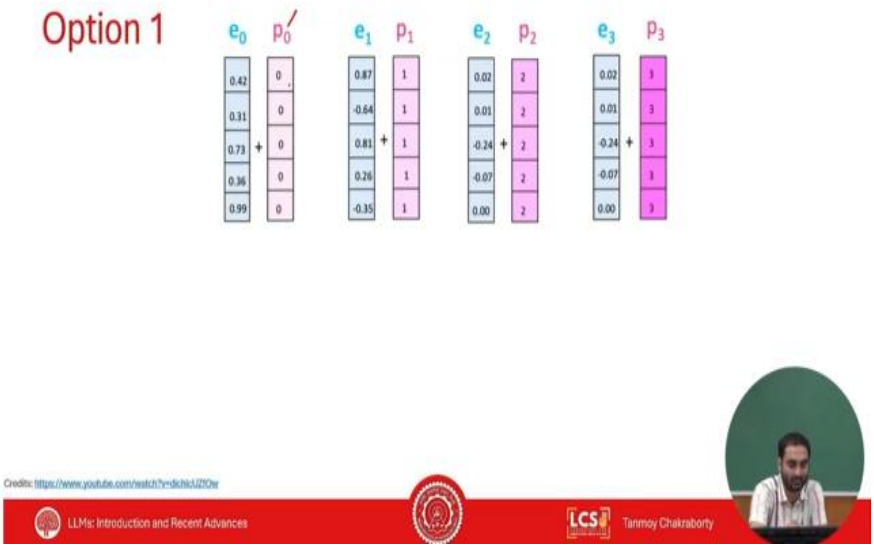
This is published in a journal, but you can look at the survey paper to know more about different types of which are encoding methods. So in transformer, we have already seen that for every token, let's say these are three input words and these are X1, X2, X3 are the embeddings of the input words. These embeddings can come from a pre-trained model like, let's say, word2vec or glafkite.bottles, or it can be a simple 1-0 encoding. And what we do next, since these tokens are accessed in parallel, just to give more information about the position, what we do, we will add a positional encoding step here.

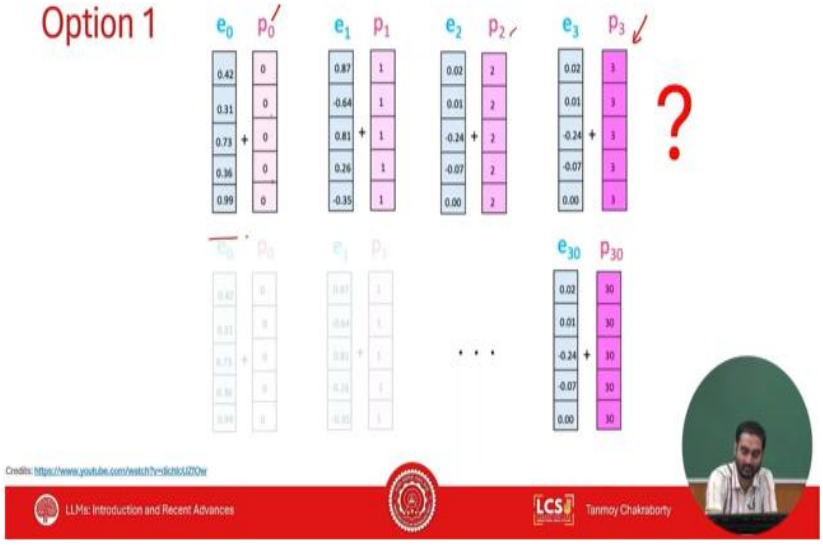
Now, this is a vector, a positional encoding, a positional embedding is a vector, right of same size of the input If the input vector size is four this will also be four right So this is the positional encoding embedding for token one this is took for token two for token three and so on and so forth and then what we do? We will basically sum the word embedding right the token embedding and the position encoding and this will give you the resultant embedding which is position ever okay Now obviously, you may ask why some, why not concatenation? There is no reason behind, concrete reason behind this, but I mean you can also concatenate the original embedding vector and the positional embedding vector. When we sum them up, the number of parameters that is needed will be decreased of course, because you are not increasing it to 8 bits, it is still a 4 bit, a 4 size vector. But there is no concrete reason you can also concatenate if you want. So, these are the position aware embedding and these embeddings will be moved, will be fed to the first layer of the encoder or the first layer of the decoder. Each layer I mentioned already, each encoder layer consists of unmasked multi-headed self-attention, a feed-forward network, and there are some components here and there.

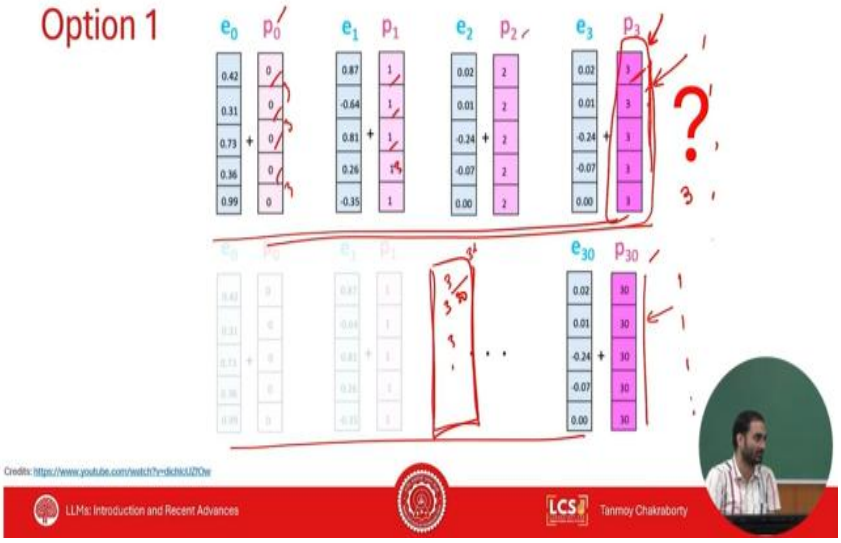
Each block of a decoder consists of a masked attention, masked self-attention, a crossed attention, and a feed-forward network. So the input that is given to the block, the first block, is position-aware embedding. Now remember sometimes people think that maybe this position information is added to every input of the layer. It's not like that. Say if this is the second layer, it is not the case that I will again add the position information here.

The position information will only be added at the beginning. Here and here. So what is the simple way of encoding the position information? What do you think? A simple way could be that you can just encode first position by one, second position by two, third

position by three, and so on and so forth. Let's say there are 30 tokens, first position will be encoded using this one, the number one, the second one is two, three, and so on and so forth. This is a simple option.







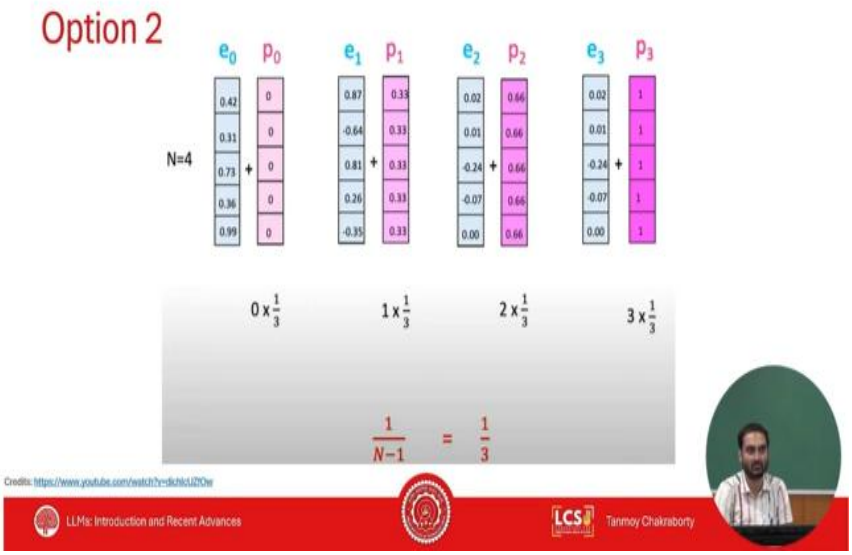
For position 0, the position embedding can be all 0s in the vector. For position 1, all 1s. For 2, all 2s and so on and so forth. For 3, all 3 and so on. This is not as simple.

There are many problems. The first problem is that the size of the input right varies. The size of a sentence varies. Let's say in one sentence there are 10 tokens 10 words in another sentence there are 20 words right 20 tokens and so on so how do you identify the max number? Right because remember the number the maximum number the maximum number of tokens should be predefined right. So how do you identify the max number because you don't have access to that.

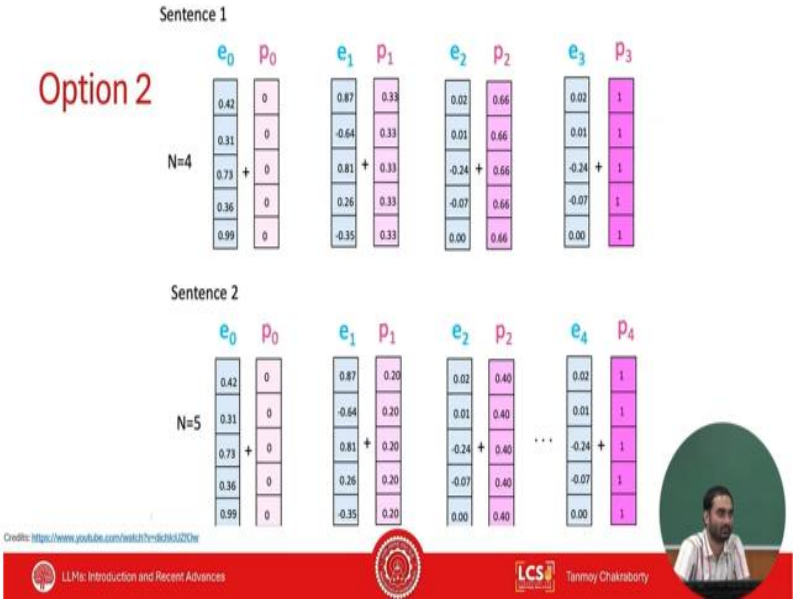
And it will also change if we increase the training set. The second problem is the last, let's say, there is a sentence where there are only three tokens. The last token embedding is 3333, the positional embedding. There is another sequence where the last token is at the 30th position. So the last positional embedding is 30, 30, 30, 30.

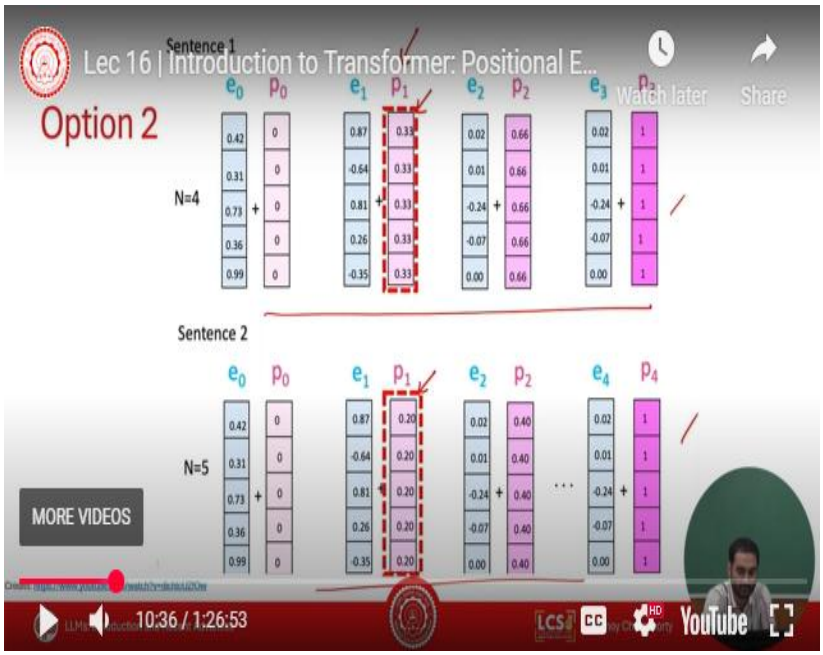
So, look at here. So, the 3 and this 3 vector and the 30 vectors are different. So, one may the model misinterpret by thinking that this 3 position embedding is same as here which is also the 3, the third position. How does the model know that this is the end of the token, end of the sequence? How does the model know this is the end of the sequence? Because here also you have 3, 3, 3. So, number one determining the length maximum length of a sequence is not possible through this approach right.

So, what is the simple remedy to address this problem? The simple remedy is you just normalize it, right? So let's say the max. So in this sequence, the maximum number of tokens is let's say three, right? So you divide everything by three, isn't it? If you divide it, then the last embedding is going to be one, one, one, one. Here also one, one, one, one. You divide everything. So here in this case, you divide by 30.



So you divide every position embedding by the max number of tokens present in the sequence, right? Like this. Okay like this. So now you guarantee that okay the last position information will be encoded by all ones irrespective of the size of the sequence length right. But there are problems here.

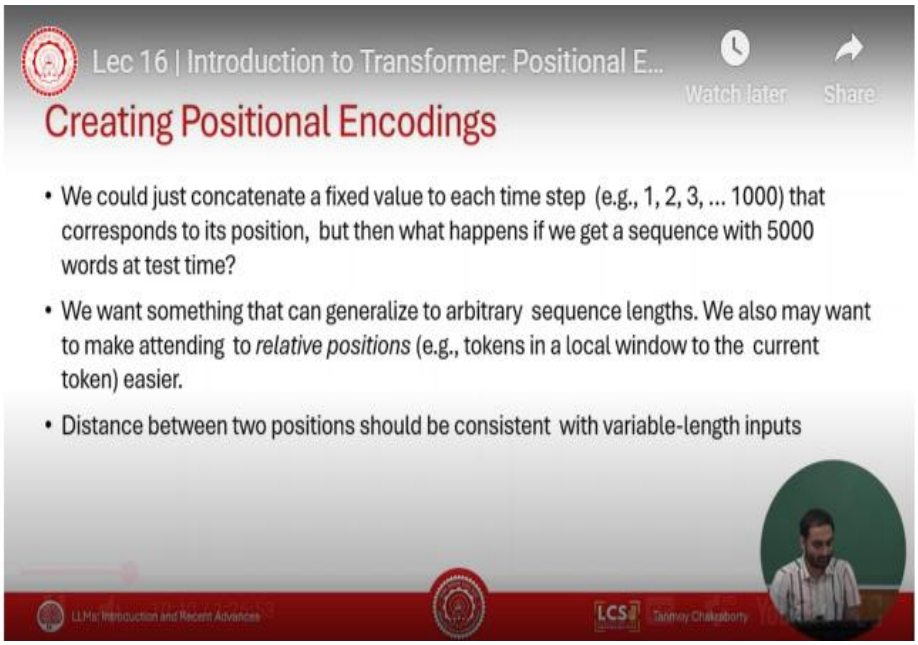


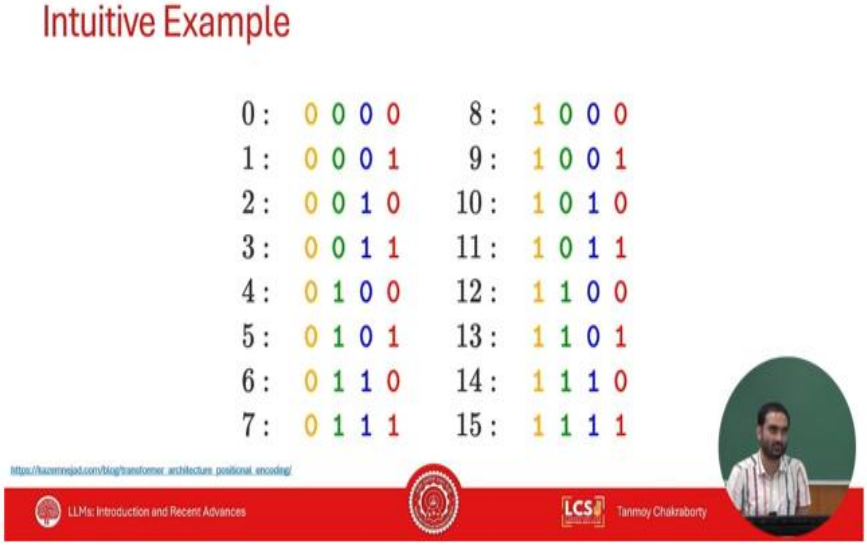


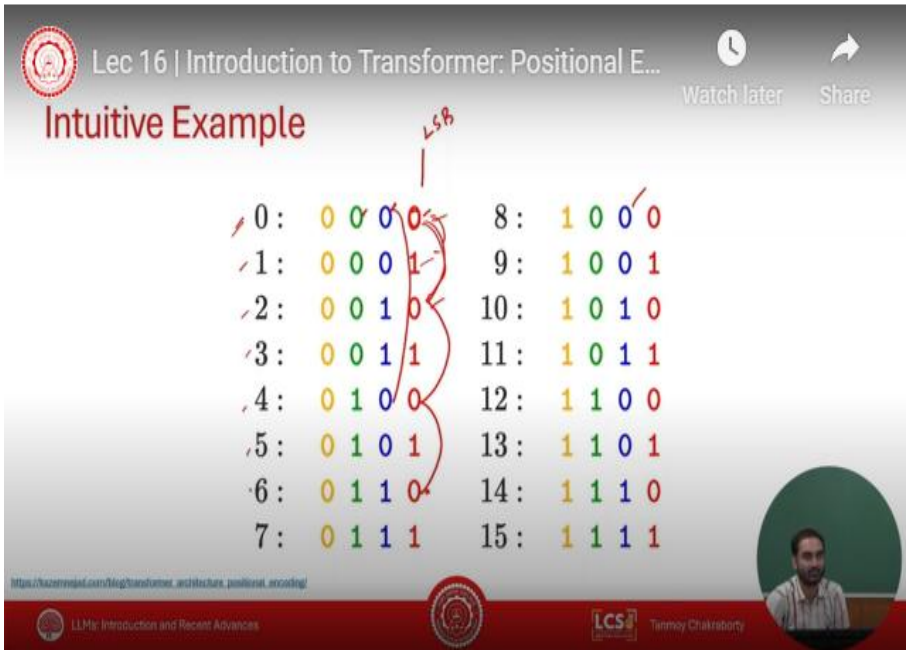
What is the problem? The problem here is it depends on the number of tokens in the sequence right so when you divide this by 3 and divide this by 30 look at the first position, 0th position, first position.

This is the position embedding 0.33, 0.33 and 0.33 right and in this case the first position is encoded by 0.2, 0.2, 0.2 although both of them denote the first position of the sequence

right. But now they are denoted using two different vectors this is not desirable right. We always assume that you know for every position there should be a unique vector. So that the model will also understand that this is a unique vector. So although this approach addresses the problem of this last token uniformity, but it doesn't address the problem of two positions should be encoded using same vectors.



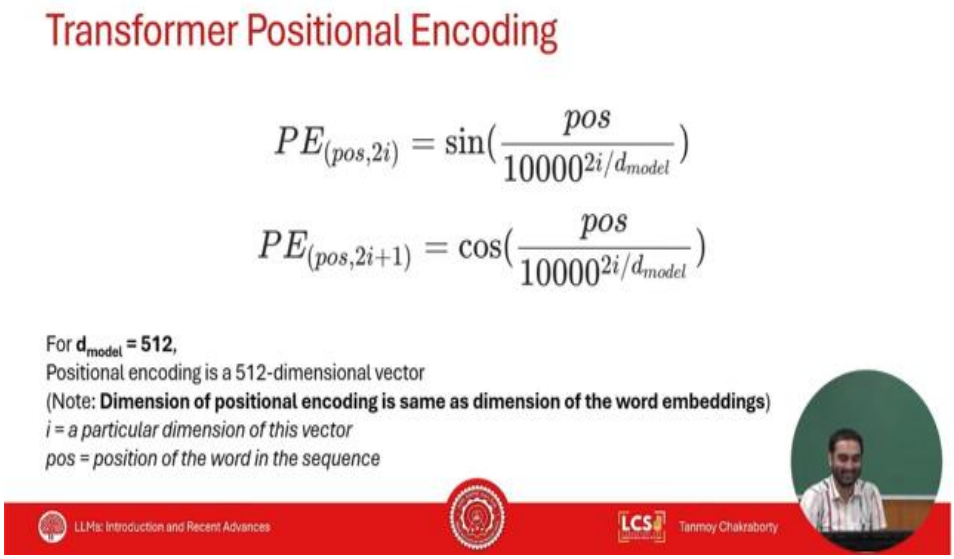




What do we do in this case? How do we address this problem? So in a simple way, the simple way to address this problem is as follows. Now this is what was proposed in the original transformer paper. Let me first motivate what does it mean. So this is position 0, position 1, position 2, 3, 4, 5, 6 and so on and so forth. Let's assume that every position is encoded by a vector of size 4.

So if I use a binary number concept, you can encode this by 0000, you can encode 1 by 0001, right, 2 by 0010 and so on and so forth. You basically convert every integer to a binary. So in this way, if you do that, you at least guarantee that all positions will be unique and the model should be able to also differentiate different positions. But of course, adding this kind of binary numbers to a word embedding, which itself is composed of fraction numbers, right? There's a 0.3, 0.1, and you add a binary vector like 0, 0, 0, 1, 0, 0, 1, 0 to a fractional vector, doesn't make much sense. It also suffers from the same problem that we don't know what is the length of the input sequence. But we can motivate, we can get motivated by this idea that if we use this kind of binary encoding techniques, you can differentiate different positions. And how do we encode this kind of thing? How do we encode this kind of representations? Now, look at this numbers or look at this vectors very carefully. Let us say this is the least significant bit LSB.

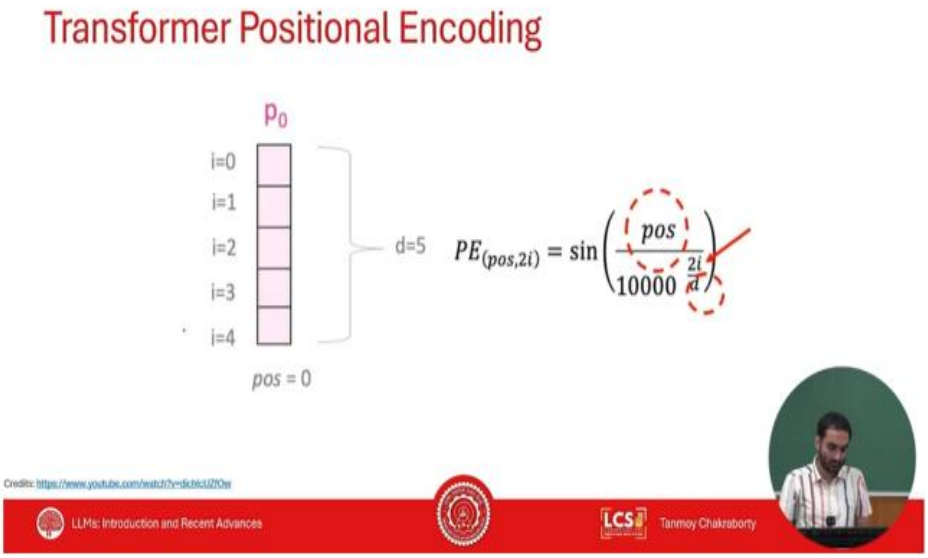
Every number, every unique number is altered every second position. 0 here, 0 here, 0, 0, 0. Every number here is altered in fourth position. So, here, here, here and so on. Every number here is altered in eighth position. Isn't it? So, you see some sort of periodicity and we know what's the best way of capturing periodic signal using sinusoidal representation.

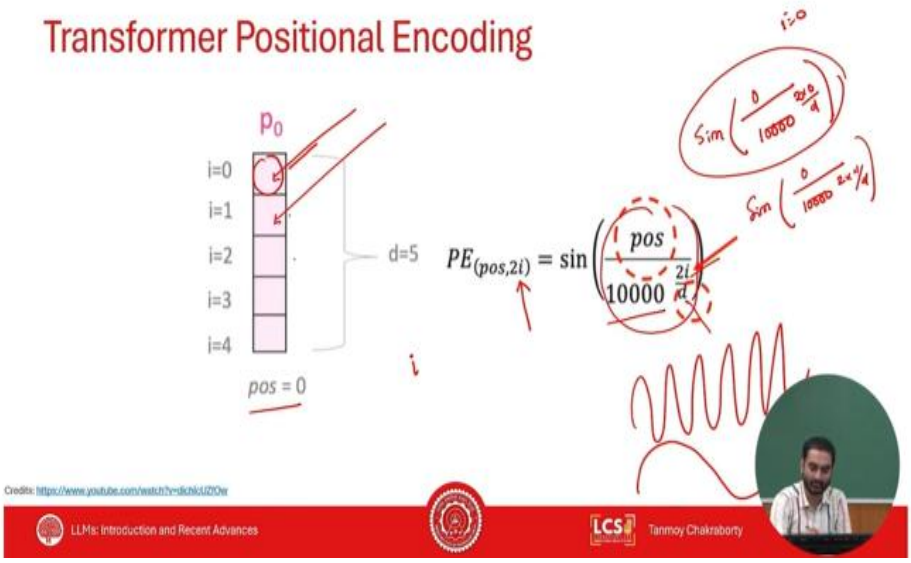


Don't get afraid by this formula, I will explain. So, this is the positional embedding formula that was proposed in the transformer model. If this is the position vector, and there are elements in this vector right. These are different elements right, first element, second element and these elements are denoted by i okay.

So i equals to 0, i equals to 1, i equals to 2, and so on and so forth Here you see these are elements okay. This is a vector and these are different elements. So instead of using 0 1 i will use some trick to essentially determine the value of every element in the vector. So, i denotes the element within the vector, pos is position, so position 1, position 2, position 3 and so on and d model is basically the model size. So, in this case the model size is 512.

Okay now the model size is already fixed is a constant right position is also fixed we know the position and i is also fixed and we see how we can use this.





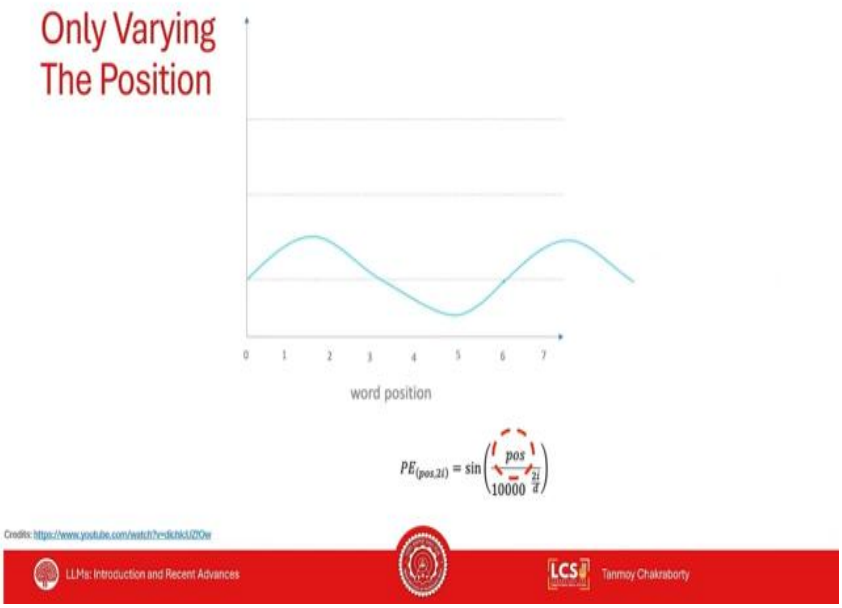
Okay so let's assume that i would like to encode the position information of the first position so pause is 0 okay and the size here is 4 okay. So this i will vary from 0 to 4. This is i equals to 0, i equals to 1, 2, 3, 4. So every element of this position vector will be determined by this formula.

So in this case, this is going to be sine position equals to 0 by 10,000 to the power, let's say you are interested in i equals to 0. i equals to 0. 0 by d is whatever right d is a fixed number.

So, this number is going to be the first element of the position embedding ok. The second element of the position embedding is going to be right sin 0 10000 2 times i 2 times 1 by d.

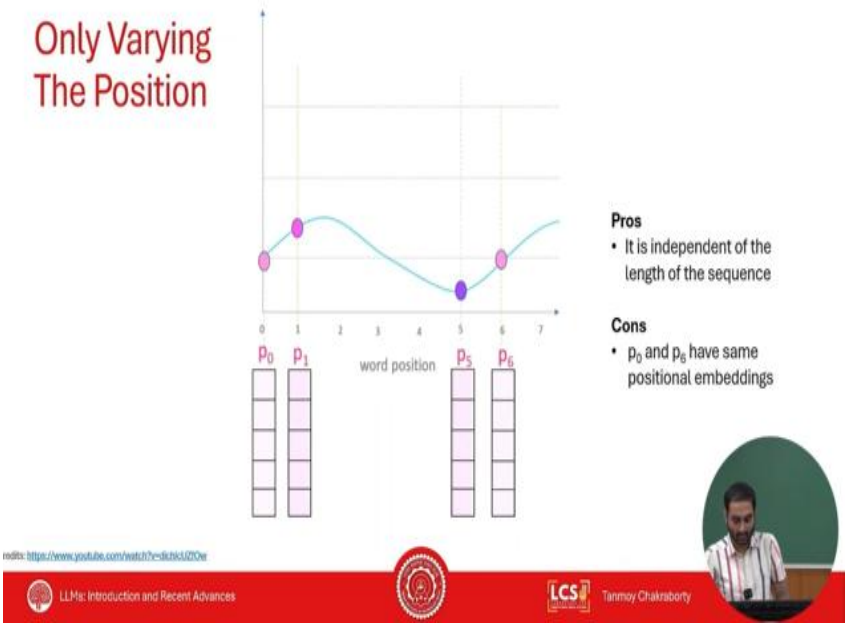
You may ask why this is the number 10,000 coming here, right. This is just a number that they proposed. And this determines, this value determines the frequency of the sine wave, isn't it? Think about it.

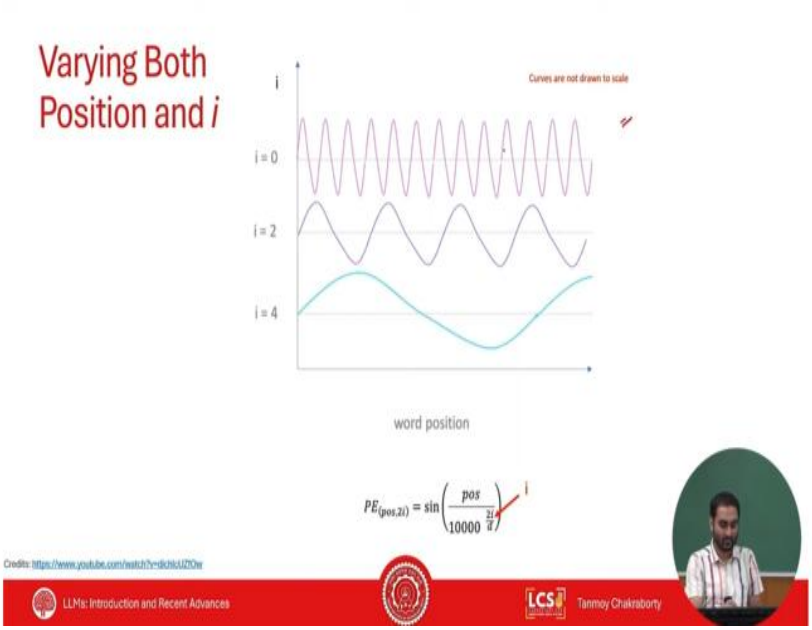
D is fixed. D is fixed. If i equals to 0 or i equals to 1, i equals to, i is a less, I mean a small number, then the frequency will be high. Right if i is a large number then the frequency will be less and we know that a high frequency sine curve looks like this and the low frequency sine curve may look like this right. So we capture the first or the upper part of the vectors through the high frequency sine wave and the lower part of the vector by the low frequency. I mean upper part means i equals to 0, i equals to 1 and so on. So more or less you understood the concept, right? Now let us see.



So you may wonder why do I need this i? Let us forget this i okay and let us only embed the positional vector using only the position information okay. So basically sine position.

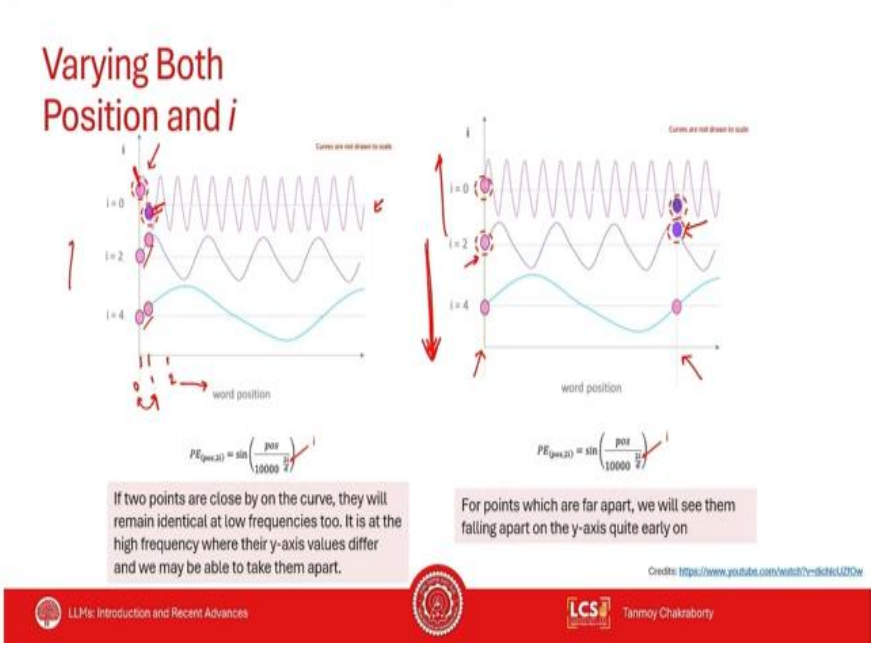
Now look at this figure here. x axis indicates the position position 0 position 1 2 3 4 and y axis is a sine value.





Look at here, if we do that the 0th position value and the 6th position value would be the same, isn't it? Similarly you will see that the first position value and the 7th position value will be the same right.

But all these positions should be unique right. Therefore we add the concept of frequency we add the concept of i index right and if we do that if we add this then it would look like this i equals to 0 high frequency right you see the high frequent wave right i equals to 2 a bit low frequent wave and i equals to 4 even lower frequent wave and so on and so forth so 0 2 4 these are different indices in the vector now The concept wise I think this is clear.



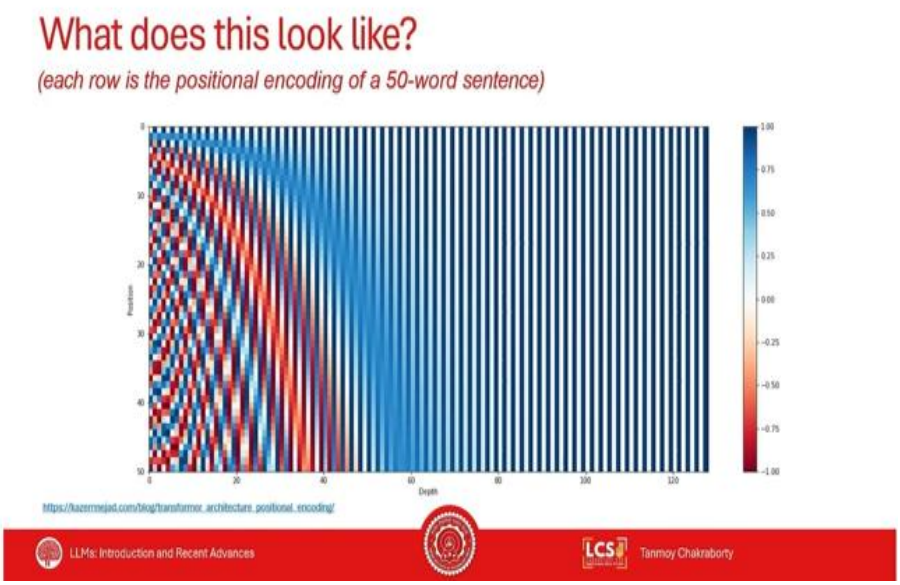
Now let's look at the beauty of this formula. So let's say there are two consecutive positions, position 1 and position 2. X axis indicates position, position 0, position 1, position 2 and so on and so forth and y axis is the sine value with different i's.

Let's look at two consecutive positions 0 and 1. Okay, ideally, what do you think? Intuitively if two words are close by right versus if two words are very far should that information be also captured in the position embedding some ways. If you look at this one carefully zero and one I will call this part of the figure as the low frequent band and this is the high frequent band. So, two nearby positions 0 and 1, you see that their low frequent sign numbers are quite similar. They will be separated by the high frequent signal.

Look at here. So, they will be separated by the high frequent signals. It means that if you compare two embeddings, position 0 embedding and position 1 embedding, you will see at the 0th position, 1 position, there are changes. But as we move away, as we move far, i

equals to 4, 6, 8, 10, 12, 30, 40, you see the numbers are more or less the same. Agreed so two nearby positions are mostly separated by high frequent bands, high frequent signals and high frequent signals are captured by the first few elements of the vector i equals to 0 i equals to 1 and so on and so forth. Okay now as you move on let's compare the zeroth position and let's say sixth position right, zeroth position on the sixth position.

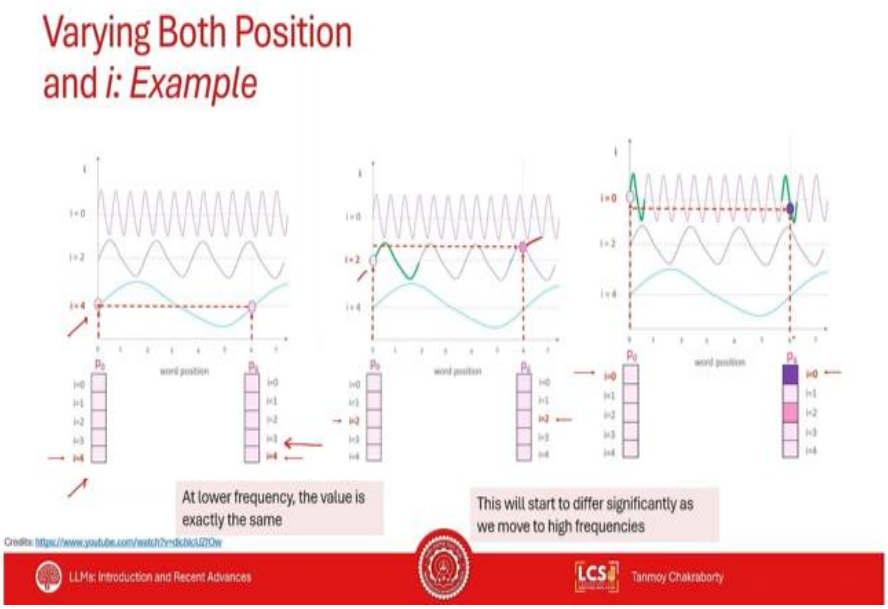
You see that the low band the low frequency signals they are itself capable enough to differentiate them. Look at this one. So here you see the difference mostly the difference here but here you will see the difference at the low frequent zone okay. So when you compare 0th vector 0th position vector and the 6th position vector you see that the high frequent regions will of course differentiate them but the low frequent regions will also differentiate.



The best way to visualize this i have a visualization. This is a visualization So this is an position embedding of 50 words okay 50 positions. So each row indicates a vector position vector right. Each row indicates a position vector. So the first row is the position information of the zeroth token and second one is the first token and so on and so forth.

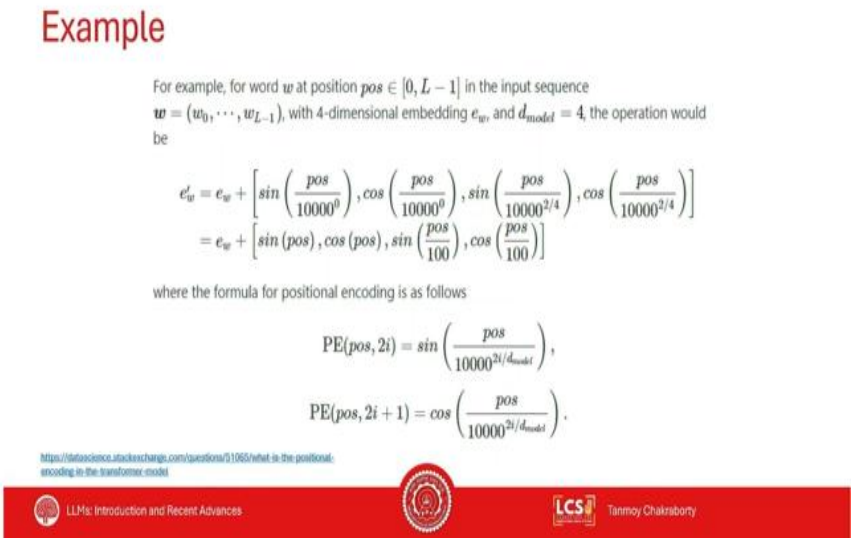
Each column indicates the elements of the vector i. Look at the first embedding. You see this periodicity here. Right this white block is getting repeated right every every single

position if you compare the first two bits here the first two elements here you see there's a difference and these are high frequent regions, isn't it? Because i equals to 0, i equals to 1 and so on and so forth This is not exactly the sine curve by the way because we will discuss it is composed of a mix of sine and cos cosine.



Now let's look at the third important property here. So look at here There are two positional embeddings for 0th and 6th respectively.

At i equals to 4, which is the lower frequency range, they are more or less the same. As you move on, i equals to 2, you see a difference. As you move on, as i equals to 0, you further see the difference. So at low frequency the value is exactly the same, this will start to differ significantly as we move to the high frequency zones ok.



So now the original transformer paper they used a combination of sine and cosine, right.

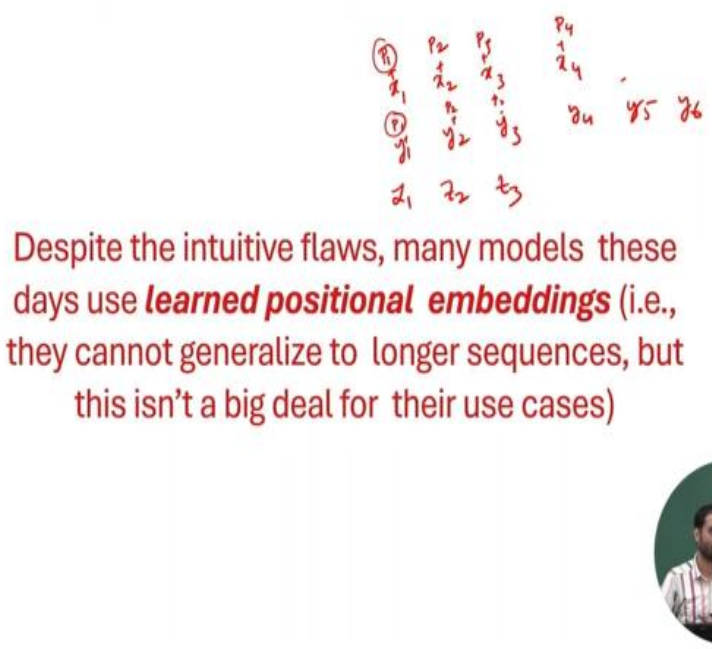
So this is the formula that they use every odd position of the vector okay let's say i equals to 1, if i equals to 1 then ei will be 2 and ei plus 1 will be 3 right. So the second element will be driven by sine and the third element will be determined by cosine the frequency is the same, this will be determined by sine and this will be determined by cosine. So this is sine and this is cosine right. If i goes to 0 then this will be sine and i equals to 0 means 2i plus 1 will be 1 right.

So this will be cosine. So the first element will be determined by sine, the second will be cosine, third by sine, fourth by cosine and so on and so forth This is even complicated. So far we have only seen one type of signal, sine. Now here they said, let's mix both sine and cosine. Both are periodic and kind of complement.

So this will even help differentiate positions further. Let us look at whether this concept is enough to satisfy all the position embedding you know properties. So, it is independent of the length of the sequence right. It will give same embedding to the same position of in different sequences right ok. Two nearby watts will be differentiate by the high frequency region right whereas words which are far will be differentiated by both high frequency and low frequency region right. So if we take the dot product or some sort of similarity, you

will see that the positional similarity between nearby words are quite high whereas those words which are far their positional similarity is quite low.

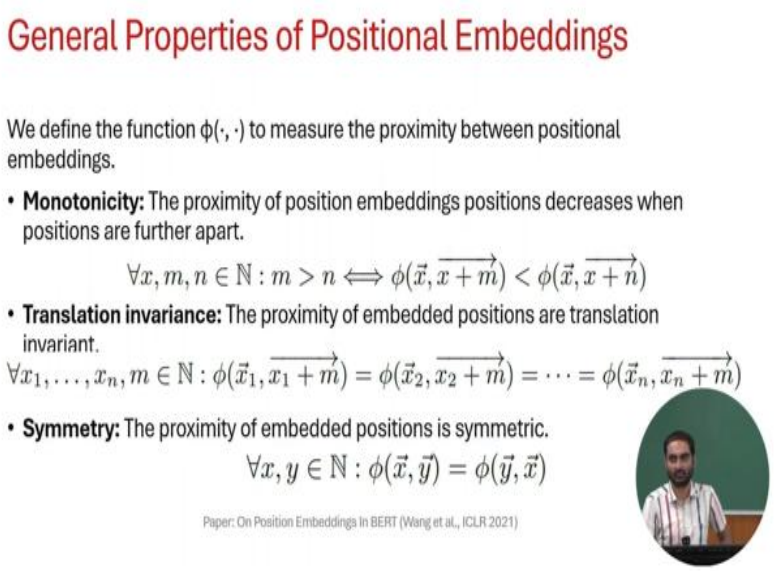
Now this is the formula, let us say the model size is 4, so d mod is 4 and you have sin cosine, sin cosine and so on and so forth. So this is the positional embedding, and this will be added to the actual word embedding and this added vector will be passed to the next layer, the first layer. This is just a visualization. You can also just run this and see how it looks like.



By the way, so this was proposed in Vaswani's paper, but then later people realized that let's not make it complicated.

Let's just consider this position embedding as a trainable parameter. Let's train this positional embeddings. So for every position you will have a trainable vector which will be learned and this position vector will be same across all the sequences that we are going to fit in. Let us say you have the first sequence x1, x2, x3, x4, second sequence y1, y2, y3, y4, y5, y6, third sequence z1, z2, z3. Okay the position vector that we will learn would look like this let's say p1 p2 p3 p4 right plus plus plus plus p1 the same position vector will be learned based on the second sequence p1 p2 p3 p4 right and so on and so forth So you can think of it as a kind of a dictionary where every element of the dictionary indicates the position vector.

So whenever you feed a sequence, you fetch the corresponding position encodings from the dictionary and you just add them and pass them. And then when you do back propagation, the position embeddings will also be updated.



So these are the three properties that we generally consider when people think of a new positional embedding techniques, right? The first property is called monotonicity, which says that the proximity of the positional embedding positions decreases when the positions are far apart. I already mentioned, right? Let's say the position X.

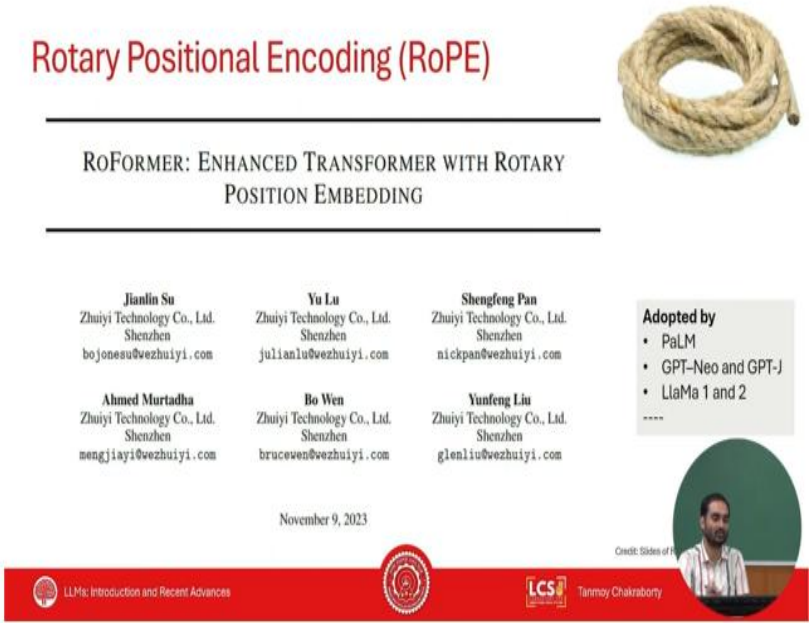
So this is the vector, right? Corresponding to the position. And there is another position X plus M. x x plus m and x plus n okay. If m is greater than n meaning n comes earlier so x x plus n x plus m right. So the proximity between this and this should be higher the difference would be lower and the proximity should be higher. If two words are close by, right they are near their proximity should be higher, okay.

So this proximity will be higher than this one Proximities can be captured using similarity and any other metric. The second property says it should be translation invariance, meaning the proximity of embedded positions are translation invariant. Now, what does it mean? Let's say x1, x2, x3.

These are three positions, x1, x2, x3. And there's a number m. So x1 plus m x2 plus m x3 plus m right. So you have three pairs now x1 x1 plus m, x2 x2 plus m, x3 x3 plus m right. If you measure the proximity they should be the same it should not depend on the the original position x1 x2 x3 these are basically highlighting the relative positions okay the

relative position should be intact The relative position difference or the proximity between 1 and 3 positions and 4 and 6 positions should be the same. Symmetricity, it is also very obvious. The proximity between x and y, two positions should be same as the proximity between y and x.

So, if you try to come up with a new positional embedding, these three properties should hold.



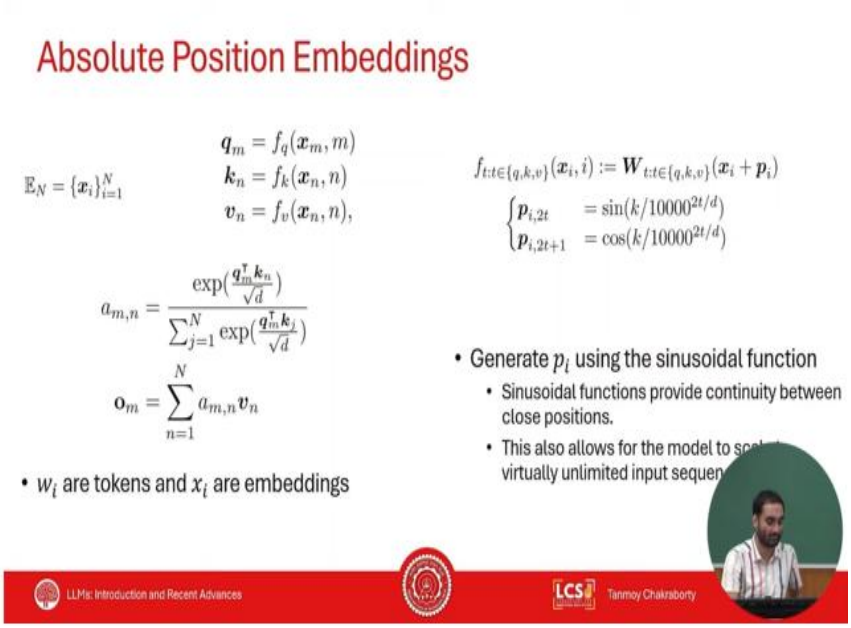
So, let us move on to the very recent one. In this paper, this is called Rotary Positional Encoding. This was uploaded in 2023 and since then, recent methods like PALM, GPT-Neo, GPT-J, LAMA-1, 2, all these models are basically using this positional encoding.

Nobody uses the sinusoidal representation. The idea behind this is a bit complicated but very different from the sinusoidal approach. So, what is the idea here? By the way, they proposed this rope embedding, rope positional embedding and they just incorporated this rope embedding to transformer and they came up with this model called Roformer. Everything is same except the embedding and that actually made a lot of difference. I will tell you why position information is so important. When you see the result, you will realize that although it looks like a very simple problem, it is not like that.

In fact, if you read the survey paper, you will see so many variants have been proposed so far with small changes here and there.

We have seen two kinds of thoughts in designing the positional embedding method. One is the absolute positional embedding. In the absolute positional embedding, you are essentially, you are only focusing on the absolute positions, right? You are trying to somehow encode those positions, right? So how do we do that? One way is either through learning, right? You basically learn all the positional embeddings, and the other way is the sine-cosine based approach that I mentioned earlier.

So this is absolute positional encoding.

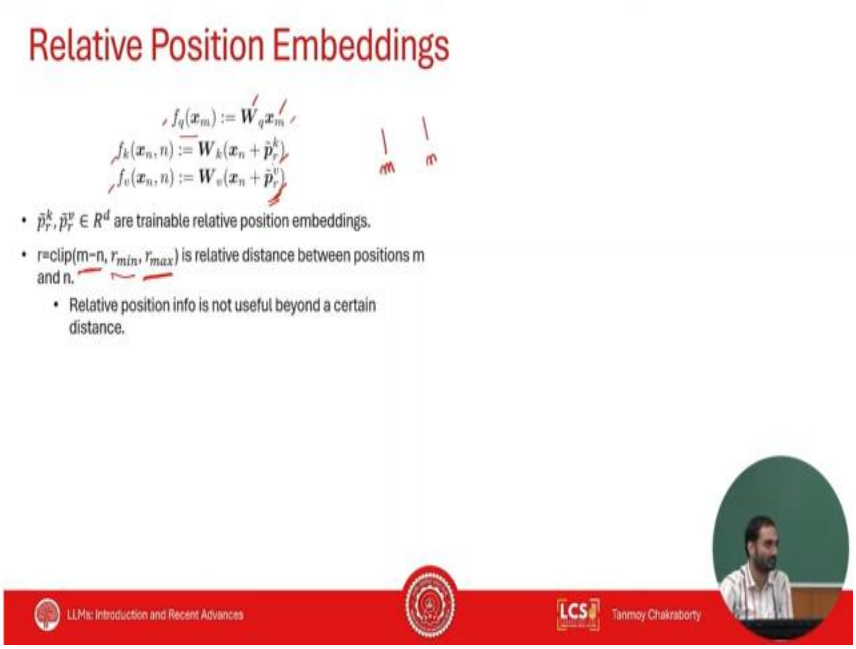


This is just a demonstration of absolute positional encoding that I discussed so far. These notations are important. Therefore, I am explaining it again. So what is this x i? x i is the word embedding, okay input word embedding for the corresponding position i or the word token i right. So what is fq? Now remember from every token i generate three vectors query key and value right query will be generated through a function some function fq right which in the normal transformer this is essentially a linear transformation you multiply the embedding with a matrix wq right we multiply this by wq and this is by wk and this is by wv.

But let's assume that these are generic functions ok. Now, these are functions of the word embedding and the position m right. In a normal transformer what we do? We basically x do x m plus p m, p m is the position information right and here x n plus p m and so on and so forth right but again let's make it generic assume that there's a function which is we applied to x m and m meaning the word embedding and the position right. Now whether this is an absolute position or the embedding or what doesn't matter then what we do once we got the query key value vectors we essentially we get the attention score right this is essentially query key dot product followed by sigmoid followed by short max and short max will give you a distribution and then your output is going to be the weighted average of the values right the short max output right and this will be multiplied with all v. Right

and in the original paper they suggested this kind of sine cosine based approaches okay so in the relative position so this is absolute position let's look at the relative position encoding in relative position the idea is that we don't care about the absolute position.

We only look at the relative position between a pair of tokens. So let's say the dog chased the pig, once upon a time the pig chased the dog right, if you look at the relative positions between these two they are the same. Right although the two words got swapped right and the few tokens you know were added at the beginning. But the relative position is the same, the relative position is i mod i minus j or mod m minus m okay.



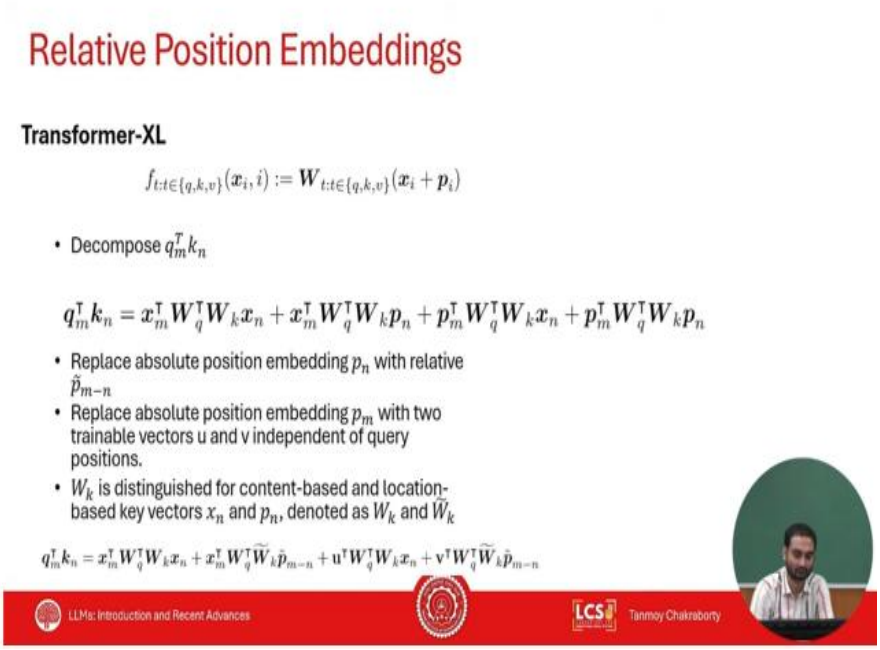
So we will basically try to preserve the relative position instead of preserving the absolute position. Let's look at some of the recent developments before rope where people mostly focused on the relative position.

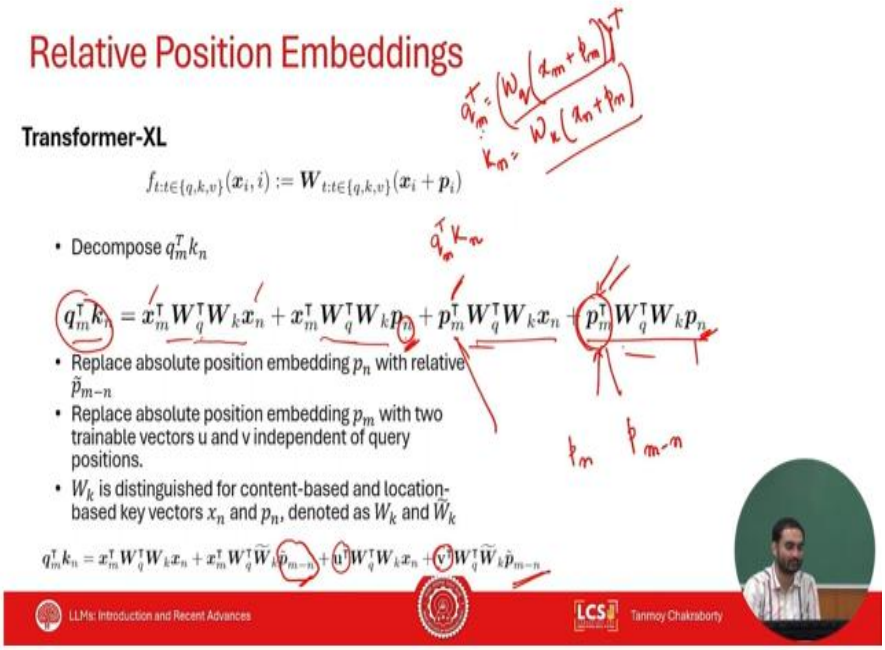
So in this paper, in this specific case, you see here this query key and value vectors are generated, but the query vector has nothing to do with the position. This is just a multiplication of the parameter and the word embedding xm. Whereas keys and values, in these two cases, they use two positional embeddings. You see the difference between this and the previous one? In the previous one, for all these query key and value cases, you basically add the position and the word embedding.

But here, only for the key and value, you add the word embedding and the position embedding. But look at this position embedding very carefully. It is PR. right not p m p n By the way, the index m indicates the query and the index n indicates the key and value. Remember the key and values vectors are generated from the same token right.

So you have a query and keys and values. So this is r. What is this r? The r is essentially the relative distance between m and n okay r is m minus n. Now if it is arbitrarily large m minus n you clip using basically threshold called r max. If it is very small again you clip using this threshold r min. So the r is essentially a clip, right of m minus n, if m minus n ranges between R min and R max then it is okay.

If it exceeds R max you consider R max, if it is below R min you consider R min. So, this is Pr. Now, how do we measure this Pr? This is trainable. You will learn this Pr. This Pr is added with the value and key, not with the query.





Okay this is one way of preserving the relative position. The second paper Transformer-XL they said that let's look at it in a different way So, ultimately, we will do query key dot product, right, query qm, key know, okay.

Ultimately, we will do dot product right. In the original transformer paper, what was q m? q m was w q x m plus p m right what was k n? w k x n plus p n. Right let us just unfold this. So when we take the dot plot of these and this, you are basically multiplying these on this right and transpose, this is a transpose because q q transpose k is basically, a plus b into c plus d okay, and this will result in four elements.

So these are the four elements okay. Now let's look at each of these elements very carefully. The first element doesn't have any position information. These are just you know what embeddings of query and key, the second element has a position information of the key, the third element has a position information of the query, and the last element doesn't have any word embedding. These are just two positions that's all okay. So what they suggested is that this n, which is the position of the key or value right.

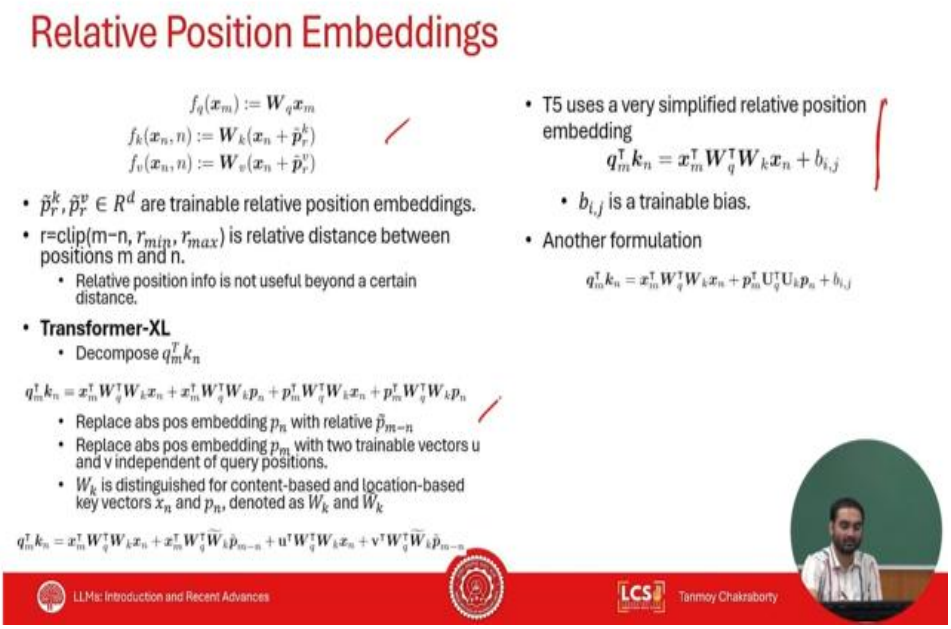
The position of key and value, let's replace this by a relative position encoding. So, I am replacing Pn by Pm-n and this I will okay so whenever you see p n it is replaced by p m minus n p n it is replaced by p m minus n okay. What's the second change? The second

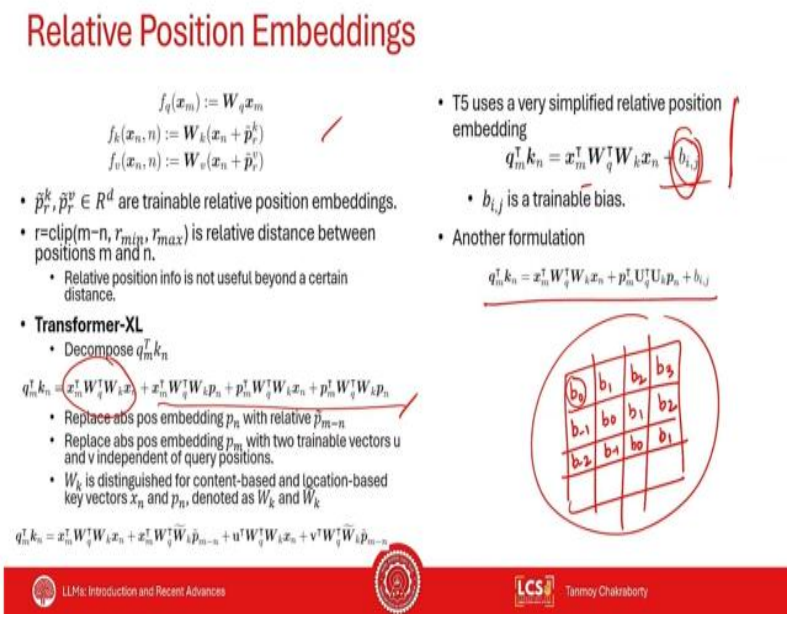
change what they said is that the, look at this one this p m, what is p m? PM is the position of the query. They said that we don't care about the position of the query because we have already taken care of the relative position of query and key. Now let's replace this by some sort of bias term or some sort of parameter that we'll okay.

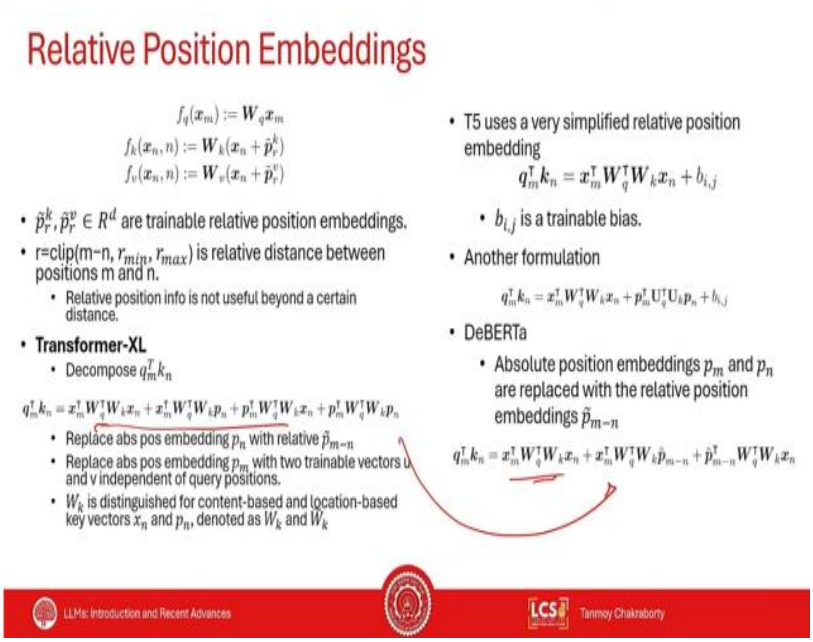
So PM you see the PM is replaced by V. So PM is replaced by V but where are this PM basically appearing? PM appears here PM also appears here right. So here now these two PMs when they get multiplied with the remaining part they'll produce something else right. So therefore, they said let's not keep the same vector.

Let's introduce two vectors. So this is V and this is U because their context are different. So you see V transpose and U transpose here. So what are the changes? Again, the changes are as follows. The position of the key will be replaced by the relative position.

The position of the query will be trainable. This is a Transformer-XL paper.





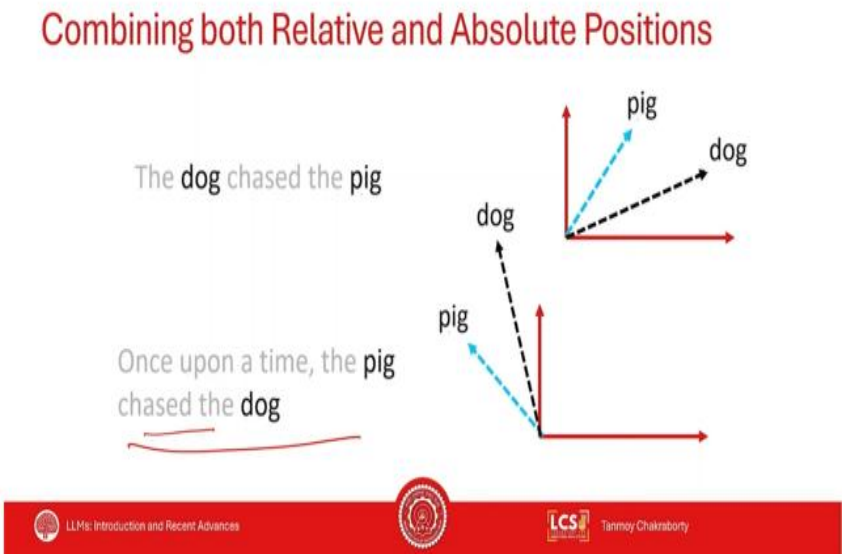


Now the T5 paper, this and this I already discussed. The T5 paper, the famous encoder decoder paper that we'll discuss later. In the T5 paper, they said, forget about all the terms. We'll learn everything. Okay will only keep that term which doesn't have any position right and the relative and the other terms just remove all the other terms where the position information is there, we will learn everything right.

And what they learn? They learn something called bij, what is bij? bij looks like this looks like very interesting So this is b0, this is b1, I will tell you what are they, this is b2, this is b3, this is b0, this is b-1, this is b1, this is b2, this is b-2, b-1, b0, b1 and so on and so forth. What are these numbers? This B0 is the bias or the learnable parameter which indicates basically the relative position between a current position with itself. Right three with three, four with four. What is b one? b one indicates the distance between three and four or seven or eight one or two right two words two subsequent what is b two? Three five right eleven thirteen.

What is b three, three words. What is b minus one, okay two one. Therefore, this is bij okay and this again this will be learned and this will just be added to the query key dot product okay. There is an another formulation. In this formulation they said that it's very difficult to learn all these things let us keep this term and maybe let us keep let's say this term and the other terms can be captured using this bias and so on. DeBerta paper they again modified this one. So this was the original equation you see they further modified they kept some terms right these terms some terms are kept some terms are removed some terms are modified and so on.

So these are basically the concept of relative position encoding.

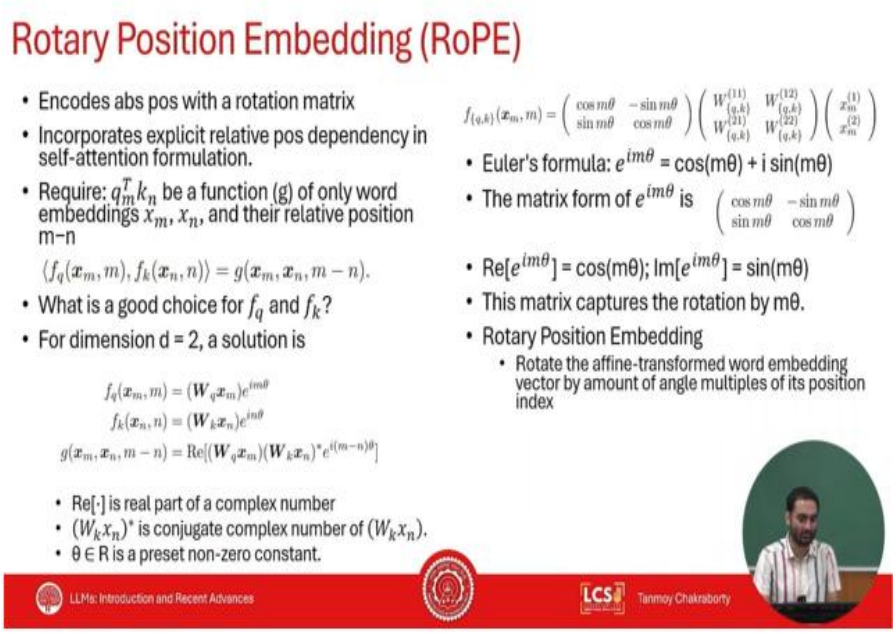


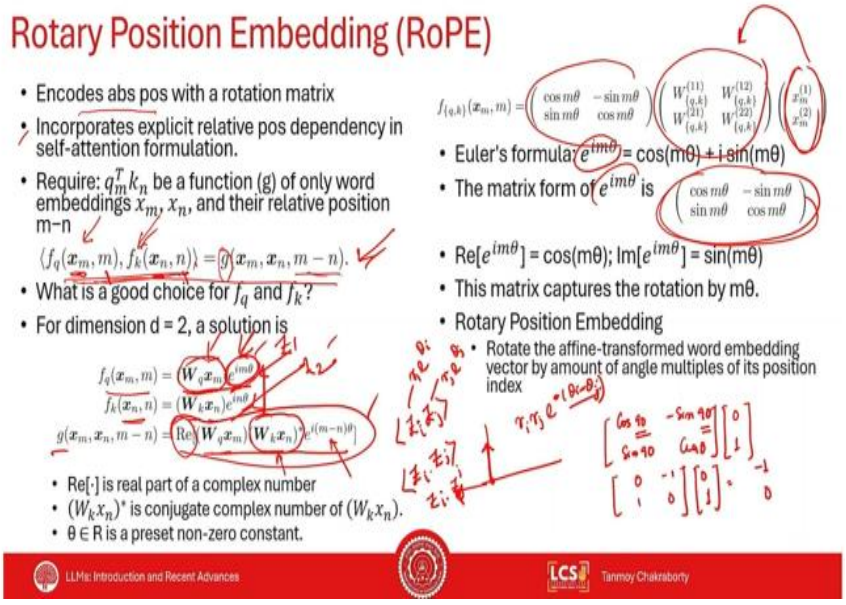
Now we will see how these concepts are used to essentially build the rope embedding okay. So let's see how we can actually combine the relative position and the absolute position information together. Rope is the first paper which essentially leverages both the absolute position encoding and the relative position encoding together. What is the idea? The idea here is very interesting.

The idea here is that the dog chased the pig and the pig chased the dog. The relative positions of dog and pig is the same. And how do we capture this? We capture this using the angle of these two vectors.

So dog is an embedding. It is basically a vector. Pig again has an embedding which is a vector. We look at this angle. If the absolute position of these two vectors will change, keeping the relative position the same, the embedding, these two embeddings will just rotate. Let us say this is the sentence but the absolute position of pig and dog is different from the previous one, right? But the relative position is still the same. They are separated by two words, right? So we rotate this to, right? This rotation essentially takes care of the change in the absolute position and the angle takes care of the relative position. Since their relative positions are the same, the angle will not change, but due to the rotation, we will also capture the absolute position change.

Beautiful idea, right? In this paper, they essentially mapped all the embeddings to the complex space. How many of you remember the complex number? Wonderful. So please be careful.





This is a bit complicated to understand.

I will try to, it's a very math heavy paper. I will not go into the very details of the math. I strongly suggest you to read the paper, but I'll try to give you the intuition behind the rope embedding. So rope encodes the absolute position with a rotation matrix. So how do you rotate a vector? We wrote out a vector using a rotation matrix. By the way, all of you should know that when we multiply a matrix with a vector, it is just a linear transformation, right?

It's just a transformation. Now, and due to the transformation, it may happen that, you know, the magnitude of this vector gets changed, right? But rotation matrix is a special matrix.

If we multiply it with a vector, the magnitude will remain the same. but the angle will change. Just to verify, you take this matrix cos 90 minus sin 90 cos 90.

Essentially, this is 0, minus 1, 1, 0. Okay. This is the rotation matrix. I claim that this is a rotation matrix. If you multiply this, let's say you apply this rotation matrix to a vector 0, 1. If you multiply, what is going to happen? Minus 1, minus 1, and 0. Right. What is minus 1, 0? Minus 1, x-axis minus 1, and y-axis 0, this one.

It basically rotated the vector by 90 degree anti-clockwise 90 degree i added 90 degree it rotated this by 90 degree okay. So we use the concept of rotational matrix. Rotational matrix is always orthogonal and so on and so there are many interesting properties okay. So we will encode the absolute position using a rotation matrix and this is important. We will incorporate explicit relative position dependency in self-attention formulation. We will see later when we calculate the self-attention formulation, the dot per between query key, we will see that there the relative position between M and L will be preserved.

You don't need to do it explicitly. Okay, if it is not clear, it will be clear very soon. Okay, now, let's look at the requirement that they that they basically considered. So what is this XM? What is XM? XM is the query embedding, right? XM is the key embedding, right? M is the query position, N is the key position, right? So they said that I will try to come up with such a formula or such a function fq fk and g which will basically follow this property. What does it mean? So, the function here which is applied to the query and the function here which is applied to the key. These are dependent on the position information, the absolute position information m and n. But this dot product of these two functions right should be same as a g another function which will be a prop which will be a function of the relative distance of two positions.

I will try to come up with such an fq fk and g where fq and fk are dependent on the absolute position but g will be dependent on the relative position and when I do the dot product between fq and fk right the resultant value would basically be is the same value which will

produce using g. And one of the options of you know which basically satisfies this is as follows. So if q this they said that if q can be written in this way. Remember what is fq? FQ is the transformation of your query embeddings and the positions right.

So, they said that we already know this one XM and WQ. XM is the word embedding and WQ is the query matrix that we learnt. We will introduce this factor this is e to the power imq the imaginary part of the vector. e to the power i, i is this imaginary part. What is m? m is the position of the query.

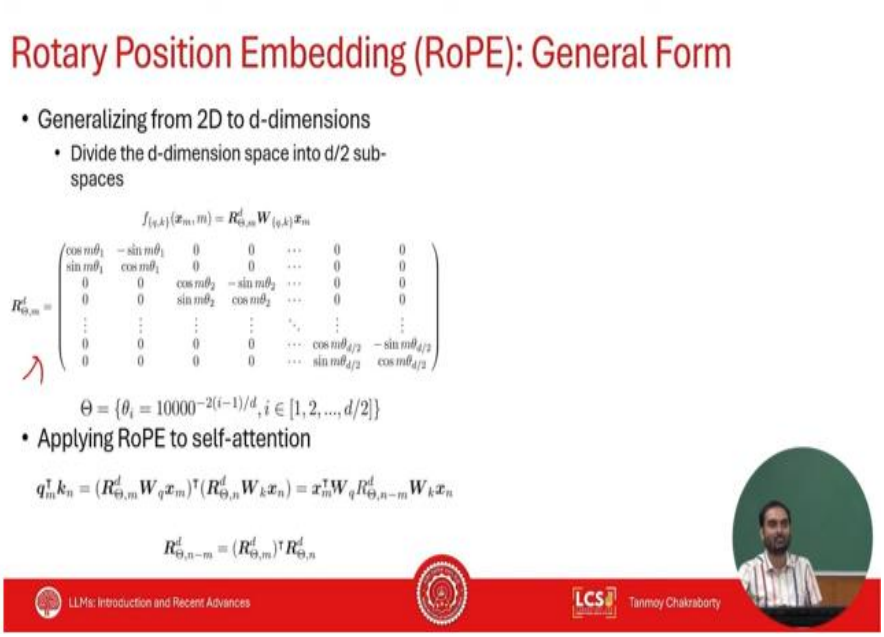
What is theta? Theta is the rotation value. So I will capture the position information using this imaginary component. okay so why this is a natural choice? I will tell you why this is a natural choice. Similarly for the key I have WK and XN right and E to the power IN theta, M theta and N theta the same rotation value theta okay. Now if you take the dot product, when we take the dot product between the two complex numbers in a polar coordinate right what will happen? Right x let's say x i x j you take the dot product right this will be x i let's say x x i is how do you represent this in a polar coordinate r theta right. So ri let's say this is r this is r i e to the power theta i and this is r j e to the power theta j.

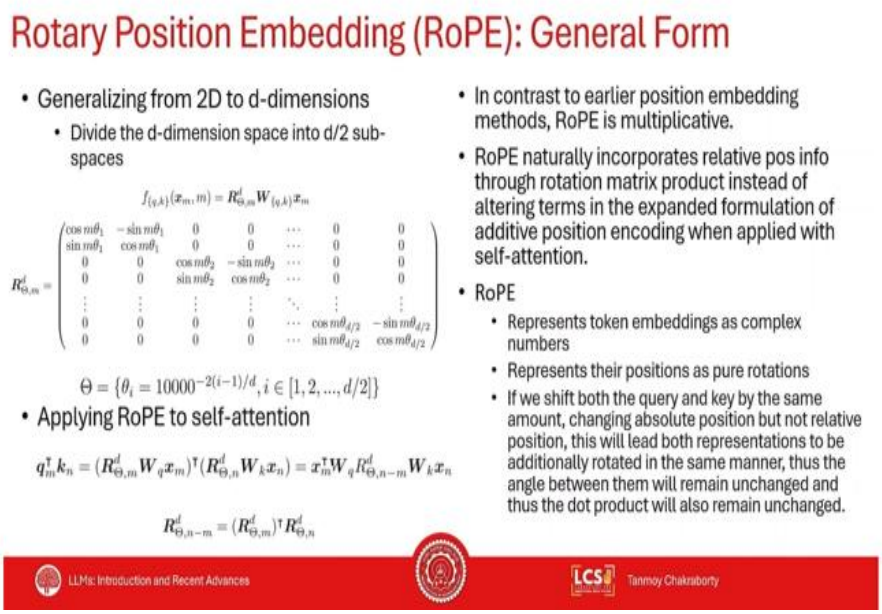
Okay when you take the dot product between e i z i z j, this is going to be dot product z i z j complex conjugate. How many of you remember complex conjugate right? Go back and check what is complex conjugate okay. when we do the complex conjugate this is essentially saying that r i r j e to the power theta i minus theta j So, if we do the dot product between these two, this will result in this one, WQXM, WKXM right this wk xn star what is this? This is the complex conjugate of this one, this is z1, this is z2, i'll have to take the complex conjugate of z2. So this is complex conjugate and e to the power im theta and e to the power im theta.

So e to the power im minus n theta okay. Now the interesting part is look at this m minus n. So now this is my g and what they said is that when we expand this right cos theta plus i sin theta right I will only take the real part of it not the imaginary part of it. So the real part is going to be the output of g. So g would look like this. Now let us see whether these three things satisfy the property that I was mentioning earlier, fq is a function of m and xm, fk is a function of xn and n, where g is a function of xm, xn and m-n. Okay. All right. Few

more maths. So this E mθ right again if you remember the complex number right E mθ can be represented using Euler's formula right which basically says that E mθ is nothing but a matrix like this. Again if you are not convinced go back and check okay. So this is E mθ is nothing but a matrix like this which is a rotation matrix.

So you are applying a rotation matrix with this transformation right. You are applying this rotation matrix with this transformation. So this is the rotation matrix, this is w matrix and this is the embedding. So when you multiply this with the linear with this weight, it will produce a vector, this vector will be multiplied with the rotation matrix right and that will essentially rotate okay. Now whatever I mentioned so far is for the two dimensional case when the size of the vector is 2.





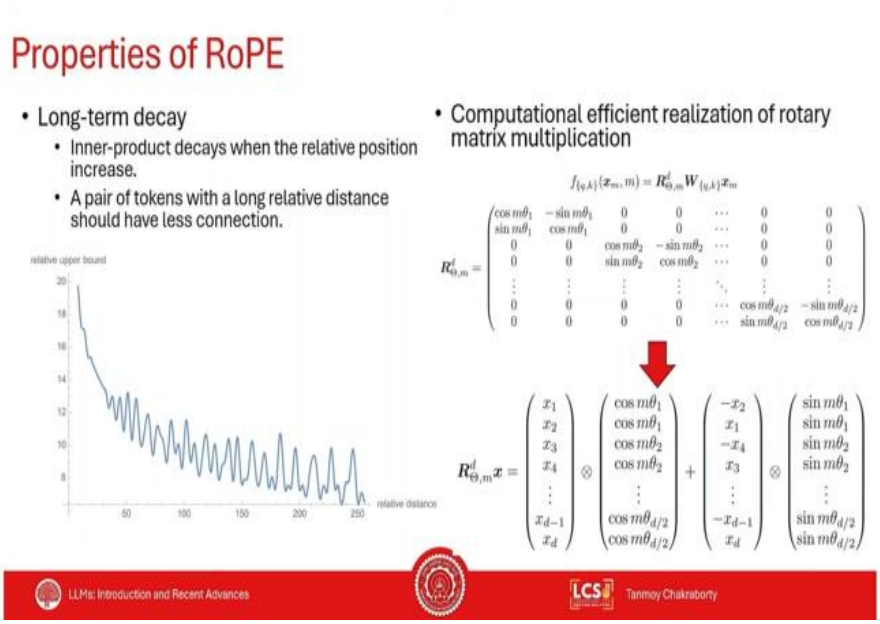
Now what happens if the vector is of n dimension? If the vector is of n dimension then this rotation matrix would look like this.

What is this? This is the d dimension. You have d by 2 number of small small blocks diagonal wise. This is the first block. Okay this is the second block, this is the third block and so on, d by two number of blocks. What does it mean, can somebody tell me what is happening here? If think about it if there is a vector of size d right, you are taking every two elements of the vector and you are rotating, you are taking every two elements of the vector and you are rotating. Remember, we also need to preserve all these properties, right? I mean the unique positional embeddings, the relative positions and so on and so forth. And how do you determine this theta? Theta is essentially same as this Vaswani's paper, right? 10,000 to the power minus 2 times i by t.

Are we learning something here? In this case, look at these numbers, all these formulas carefully. Are we learning something here? Are we learning something? We are not. Ws are already there. We are not learning the position information at all because theta is already given. Theta depends on I. Okay now when we take the dot product Qm transpose Kn right what is Qm? Qm is this rotation matrix times Wq times Xm right what is Kn? Kn is another rotation matrix right Wk Xm if you multiply if you do some arrangement here and there you will see that you will get a matrix which looks like this.

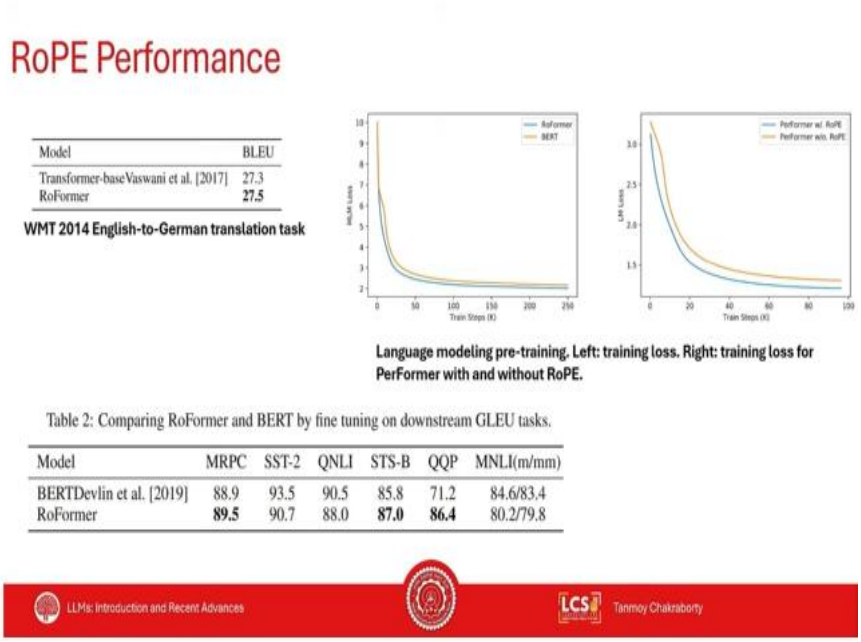
Notation matrix which looks like this right, which we call r right r is basically a function of m minus n okay. So when we do attention self-attention you basically preserve the relative information okay. So the statement that I mentioned earlier incorporates explicit relative position dependency in the self-attention formula is there. Encodes absolute position with the rotation matrix is there. So rotation matrix takes care of this m and n right m and n, when you do self attention you see the relative position is preserved okay.

This is a great paper i would say. If you read the math you will you'll be basically amazed okay. So these are some of the properties in contrast to the earlier position embedding method. Rope is multiplicative. Remember this that I forgot to mention. The position information is not additive here. It is multiplicative, right? It is multiplicative, isn't it? Multiplicative! Rope naturally incorporates relative position information through rotation matrix right instead of altering terms that I mentioned earlier and you know the rest of the things I already mentioned you can if you go through the slides you will realize okay.



Let's look at some of the again just two properties we are almost there. If I plot the relative distance between tokens 2, 3, 4, 5, 6 and the relative value of the position embedding, you see that with the increase of the relative position, the proximity value of a pair of positions, pair of position embeddings will decrease, which was one of the properties that with the increase of this, it should decrease, x plus m and x plus n, remember? The second good

thing is that although this matrix looks like a very gigantic matrix, but most of the entries in this matrix are zero. So if you do some small math here and there, if you see that this can be realized by this, where you look at the cos part, you segregate the cos and sin part, and you multiply one part of the embedding with cos, another part of the embedding with sin. This is also computationally very efficient.

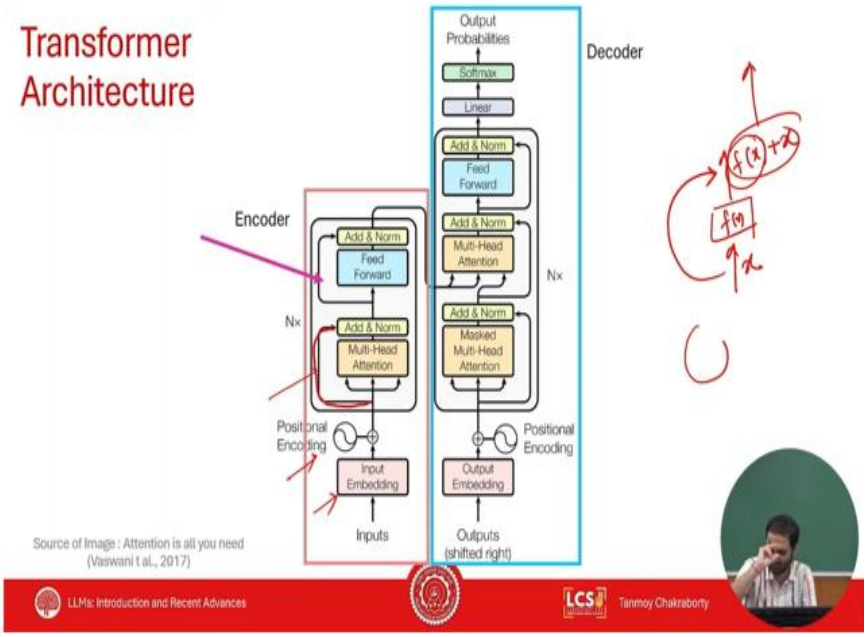


Let's look at the result. So this rope former was tested with the original transformer paper on the machine translation task, English to German. And there's a small blue score improvement. Rope former was also tested with the word model.

It showed significant improvement. Remember, we just added the positional information. Let's change the positional information. The remaining part is the same. With such change, there's a gain from 71 to 86, 85 to 87, not only that, but also in the training time. So this is the X axis indicates that the training time step, right? And Y axis indicate the loss. You see, so this orange line is the bird and the original bird without rope, right? And this blue line is the rope former.

You see that the model also converges fast, faster than bird. Okay, now if you see the, you know, there is another model called PERFORMER. And you know, there are different transformer versions, LONGFORMER, PERFORMER and so on. So in the PERFORMER

model, if we use rope, and if you do not use rope, so PERFORMER with rope, PERFORMER without rope, you see again, the loss decreases, you know, quickly compared to the case where there is the, you know, the normal positional embedding. Okay, so this is all about the rotary positional encoding part. So we have discussed what?

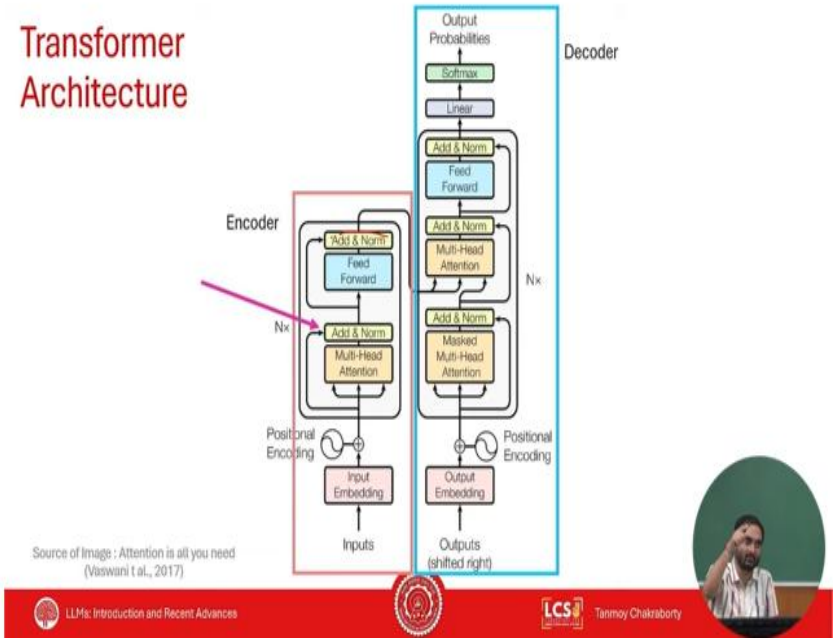


We have discussed positional encoding and the other modules are also discussed.

What are remaining? One part which is remaining is a residual connection. We all know what is a residual connection. So let us say this is a layer. This is x and this is basically a function fx. It will produce fx. So I will basically have a shortcut from input to output which will basically add the output with the input.

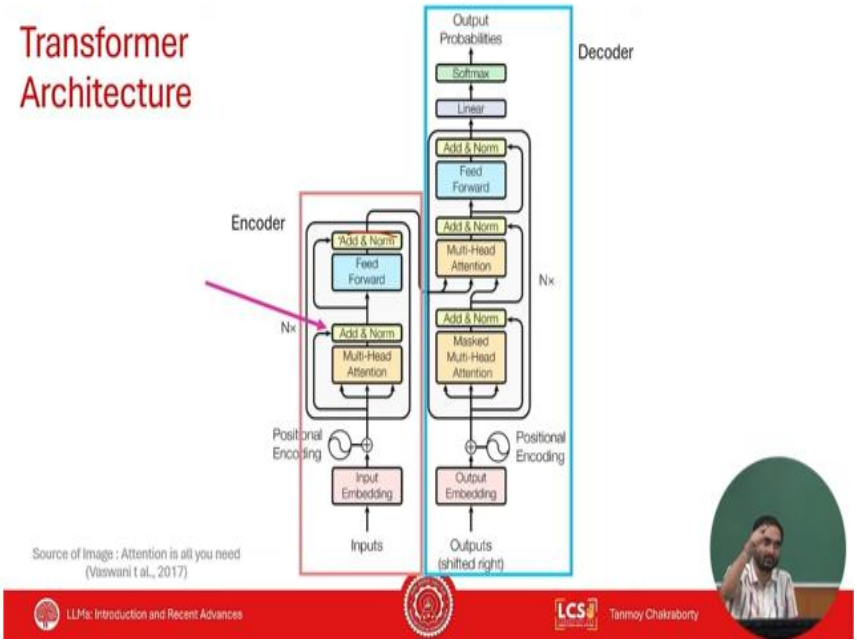
Now this will be fed to the next layer. Instead of feeding fx, I will feed fx plus x. Now why this is important? This residual connection is important because it helps you address the vanishing gradient problem because now you have a direct connection. You are not letting the gradient flow through the complicated portions of this f block. It can flow through this directly.

This will help in convergence, and there are many important properties. So what we do, we add a residual connection here across self-attention. You will add another residual connection here across this feedforward network.



The last part is this add norm. Add means the residual connection add. And what is norm? Norm is layer normalization. What is a layer normalization? when you do some complicated stuff within a layer within a specific layer, let's say the layer is a self-attention layer, you basically result in a vector right.

Now this vector sometimes due to these operations some of the properties of these vectors get changed. Now what are these properties? When you try to make the model fast in terms of convergence etcetera, we always make this vector as smooth as possible. Now what do you mean by smoothing? Basically smoothing means that I want the elements of the vector to basically follow certain distribution. Let's say you follow a distribution where you have 0 mean and units standard deviation. In that case, what we do? Whenever you obtain a vector, I basically apply another formula where I make sure that all these elements basically ranges between 0 and 1 and it also follows certain properties and so on and so forth.



Let us now look at the batch and layer normalization which is essentially one of the components of the transformer model. If you remember the architecture once again, we had input, positional encoding, multi-header self-attention, feed-forward network and

residual connection. We have this addNorm layer after every subcomponent of a layer. After every attention layer, we have this addNorm.

After every feedforward layer, we have an addNorm, and so on. Similarly, in the decoder, you have addNorm here, addNorm here. Look at here. After mask attention, you have addNorm.

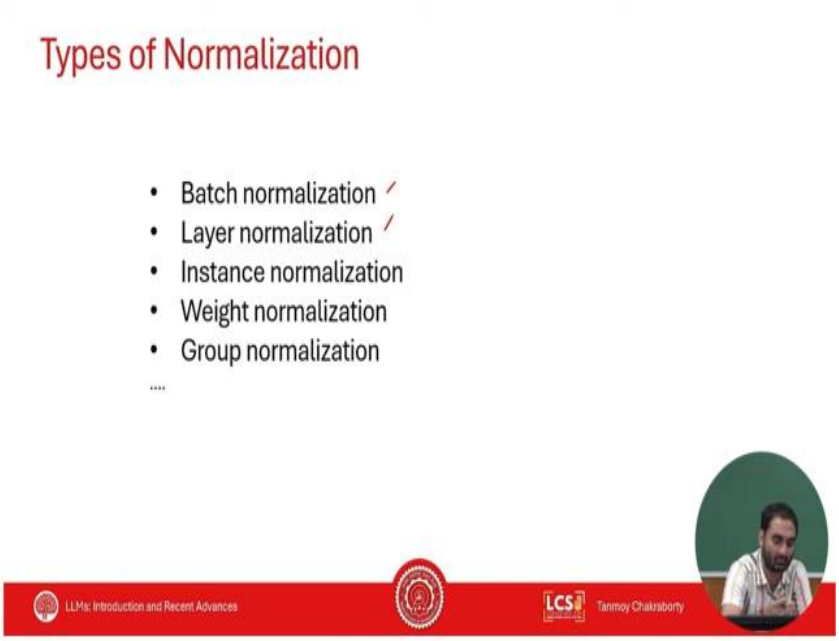
After cross attention, you have addNorm. After feedforward also, you have an addNorm. layer right. So let's look at what is the function of this addNorm layer. So add corresponds to this residual connection. You are basically adding the input to the output of the layer. Let's look at the normalization part of it. Now the normalization of a vector or of an output, it is not restricted to transformer model, it is used all over the neural networks.

Any deep learning model when you design, we always try to add this normalization layer after every block, after every layer. There are multiple reasons behind this and what is this normalization? It essentially this function this is operated on a specific vector. Now this vector can be a single vector like an output vector. In this case let us say a vector can correspond to a token or a vector can be a concatenation of multiple vectors and so on. What is the purpose of this normalization layer? The purpose is that I will try to essentially, you know, after every layer, since there are so many operations within the layer, it may happen that the entries within a single vector get deviated right and it may not start following a certain pattern.

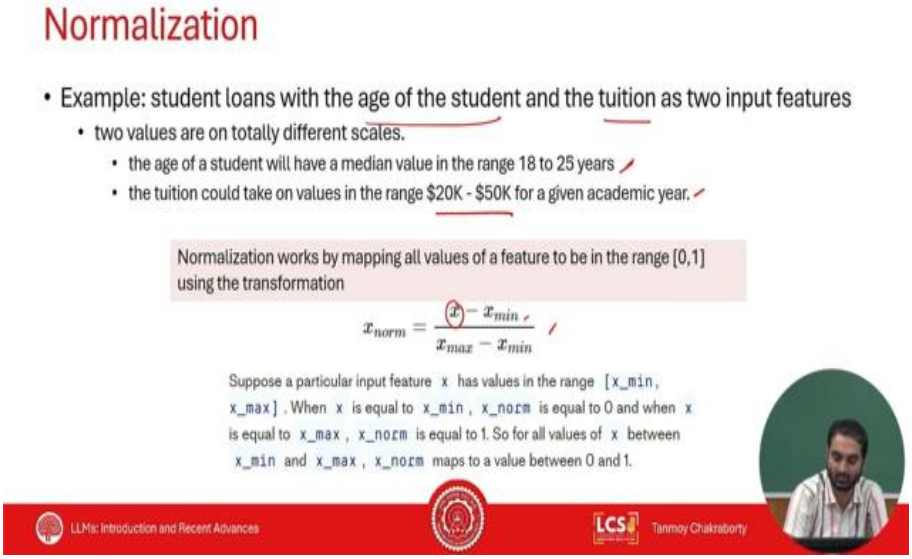
Right what do you mean by a pattern? A pattern can be let's say you want to make sure that the entry all the entries in a specific vector ranges between 0 to 1, for example. Or let's say you want to make sure that the entries actually follow a certain distribution. Now this pattern actually gets destructed gets changed when you move from one layer to another layer due to these complicated operations within a layer okay. So therefore we do this normalization. Normalization has a series of benefits. The first benefit is that if we do normalization, it will help you stabilize, it will help you speed up the training process by keeping the inputs to each layer within a similar range or within a similar distribution.

It ensures that the gradients don't explode or the gradient don't vanish during back propagation. It also is very effective in accelerating the convergence. There are empirical

studies which showed that if you use normalization versus if you do not use normalization, in the former case the model converges faster than the later case. The third advantage is preventing overfitting. There are methods like dropout some of you know in the deep learning right. In deep learning we use dropout techniques where we essentially freeze some of the weights and you know along with dropout and normalization it also helps you know address the overfitting problem empirically and essentially I mean if you think of all these three points together it essentially improves the gradient flow right across the layers.



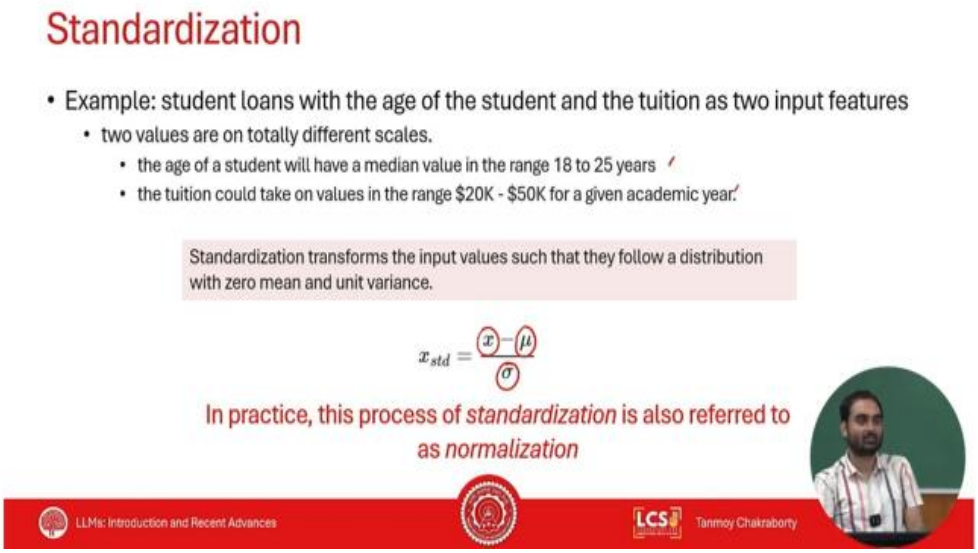
So normalizations can be of different types, batch normalization, layer normalization, instance normalization, weight normalization, group normalization, and so on and so forth. But in this specific part of the lecture, we'll focus on batch and layer normalizations. So in Transformer, we generally use layer normalization, but batch layer normalization can also be used. We'll look at the pros and cons of batch and layer normalizations.



Let's take an example to illustrate the idea of normalization. Let's say you are given an instance where there are two features. One feature is the age of a student and the other feature is the tuition fees in an academic year. So based on the age of a student and the tuition fees, you want to determine whether the student is eligible for taking a loan or not. Here the edge ranges between 18 to 25. Let's say you have a feature which ranges between 18 to 25, whereas the tuition fees in an academic year ranges between 20K to 50K dollar.

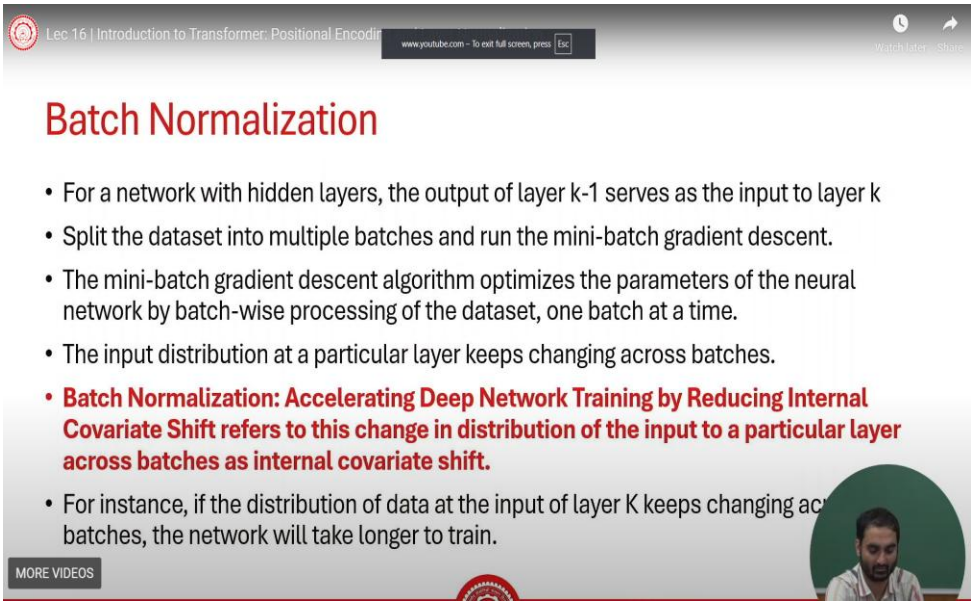
Now if you look at these two features, now these two features essentially correspond to two entries in the vector. And the ranges are very different. So one way to do this normalization is that you make sure that all these entries range between 0 to 1. How do we do that? One way of doing this thing using, you can use this kind of min-max normalization to subtract every entry by the mean entry of the feature. Let us say the feature is age, you look at the minimum age present in the training set and you subtract that minimum value of that feature from the element, from the entry and the denominator is the max minus mean, max value minus mean value.

So, let us say the max value is 25, mean value is 18, so it is essentially 25 minus 18. and it makes sure that this will always be between 0 to 1. Of course, there are other ways also to make sure that the range is between 0 to 1. Now this is called normalization.

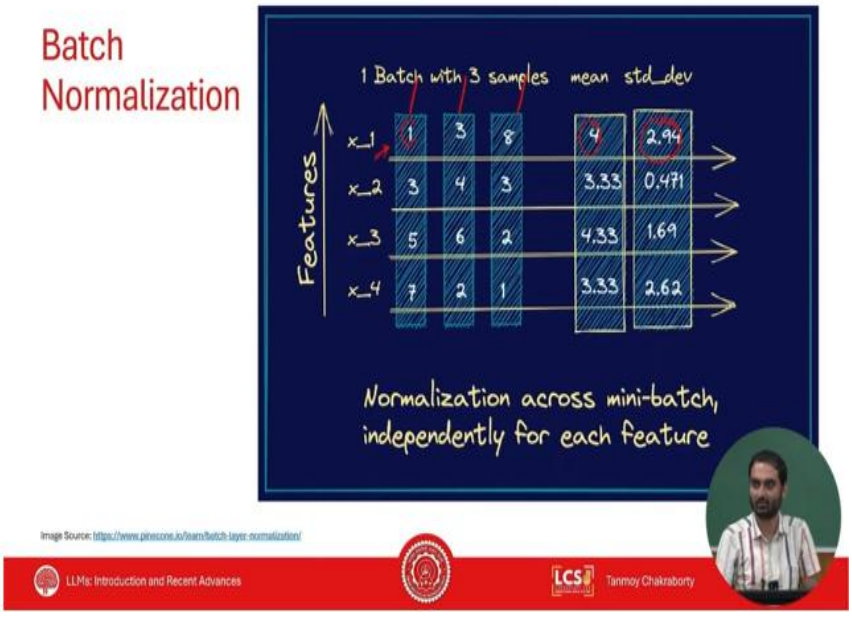


There is another term called standardization. In a standardization what we do? Again, given the two features, age and tuition fees right, what we do? We will make sure that these two features follow a certain distribution, okay.

Let us say we make sure that in this distribution the mean is 0 and the variance is 1 right. How do we do this? For every feature we calculate the mean and the standard deviation and we subtract the mean from the element, from the value and we divide it by the standard deviation. If you do that, it doesn't guarantee that it changes between 0 to 1, it will not be 0 to 1 at all, but it will follow a certain distribution where we see that the mean is going to be 0 and the standard deviation is unit. Now this is called standardization, but oftentimes we will essentially use standardization and normalization interchangeably.



Now let's focus on batch normalization. In batch normalization, what we do? We will perform the normalization operation after every batch. So we'll send a batch, we will stack all the outputs of the batch, and then we will do the normalization.

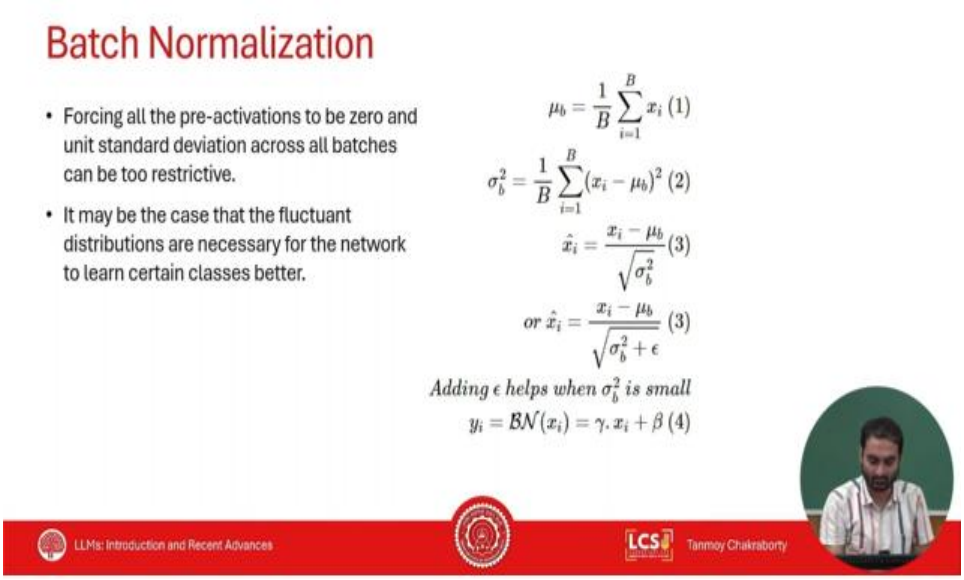


Let's first look at the batch normalization properly. Let's assume that in a batch, now batch is what? Batch is a collection of training instances.

So let us assume that in your batch, there are three instances. This is the first training instance, second training instance, and third training instance. So the column corresponds to the column indicates a feature vector or a vector. In batch normalization, what we do, you take every element of the vector. Let's say you focus on the first element.

First element of the first vector, first element of the second vector, and the first element of the third vector. You look at the first element of every vector. Then you do the standardization, meaning you calculate the mean and standard deviation, right. Let's say the mean is 4 and the standard deviation is this one, okay. Then you essentially subtract the mean from each of these elements, right, and divide this by the standard deviation. Now this is batch normalization where we are looking at the batch, all these instances of the batch and first element of that instance in the first element of every vector, right and you are essentially calculating the mean and standard deviation, right.

Now this is happening for instance, for entry one, for entry two, for entry three, entry four and so on and so forth. So you are looking at all the instances within the batch and then you are doing this operation, this is batch normalization.

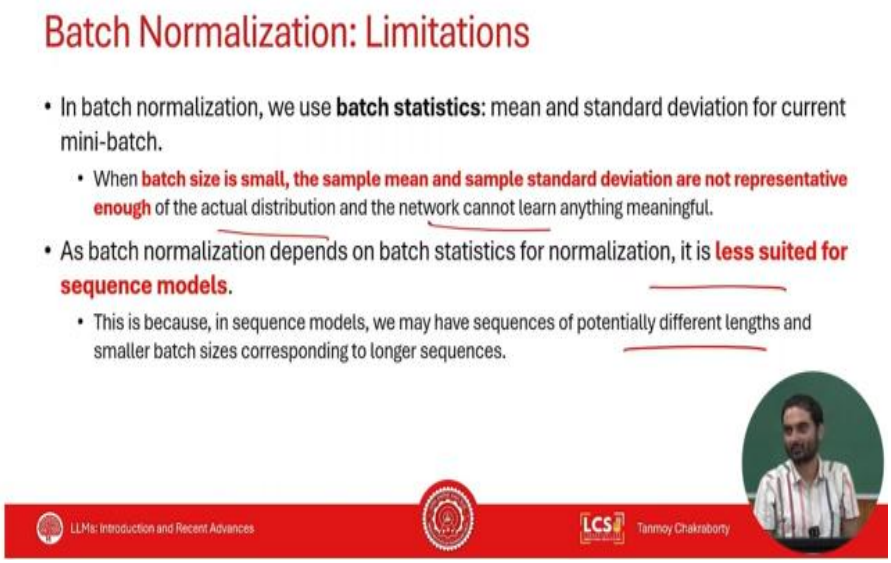


What is the problem here? The problem here is that your batch should be representative enough the batch should be representative enough of your entire population, of your entire training set. If your batch size is very small this mean and standard deviation remember the

mean and standard deviation will vary when we basically feed another batch right. So for a particular batch you have a certain mean and standard deviation when you feed another batch the mean and standard deviation will change okay.

So if the batch size is not large enough, you may not be able to basically get the correct distribution of the population. Mathematically, how do we formulate it? Look at here. There are b number of instances in the batch. okay and across the batch you are taking the mean and standard deviation right for a particular element right.

And you subtract the mean from the element and then you divide it by the standard deviation right. And this is happening for a specific element i, right. So i is the index ranging from 0 to d. d is the vector size. Oftentimes what we do? Along with this batch normalization, we also add some trainable parameters. So let's say this is the updated value of the element. Along with the updated value, we add some sort of gamma and beta, right? And this gamma and beta are basically learned during the batch propagation time. Okay?

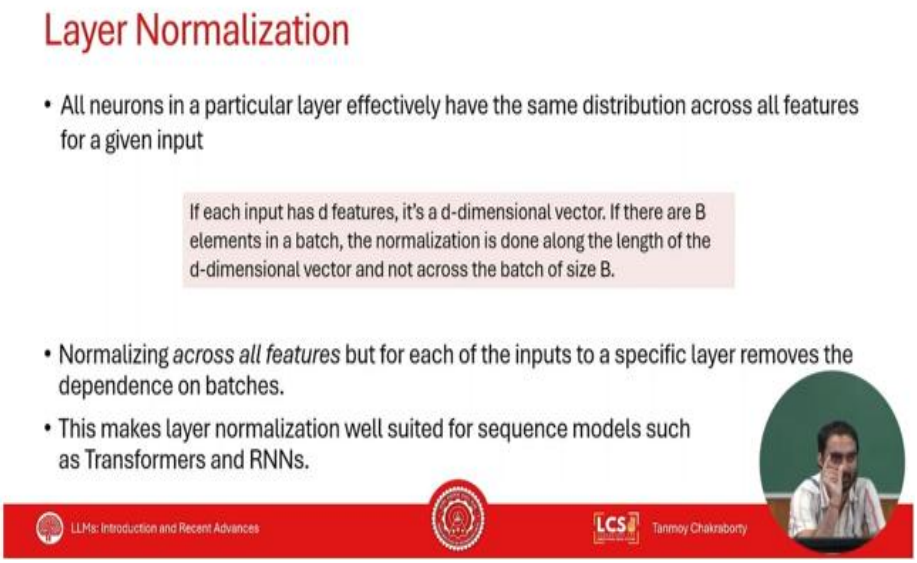


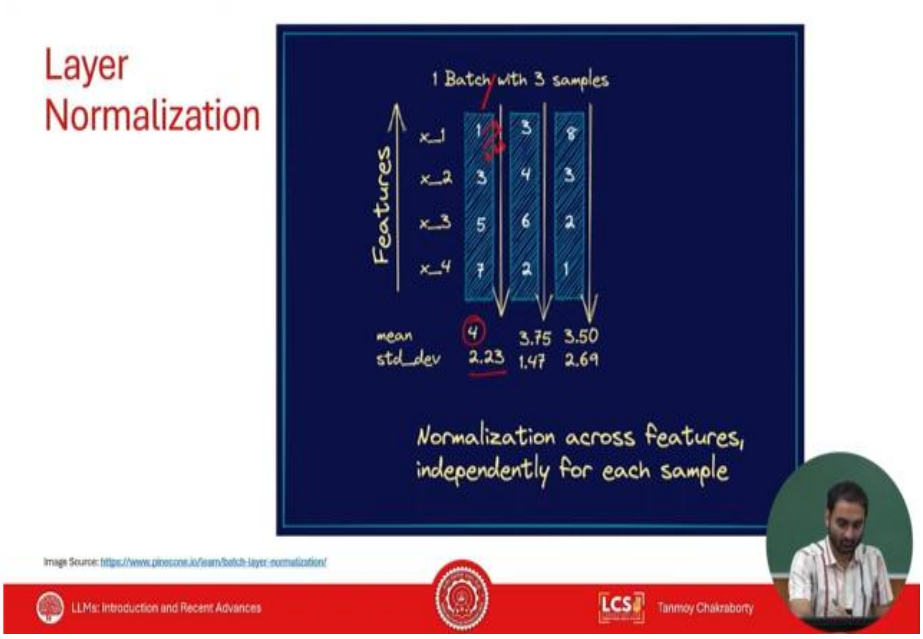
The first limitation as I mentioned, if the batch size is small, the sample mean and standard deviation are not representative enough of the actual distribution and the network may not be able to learn the actual meaning.

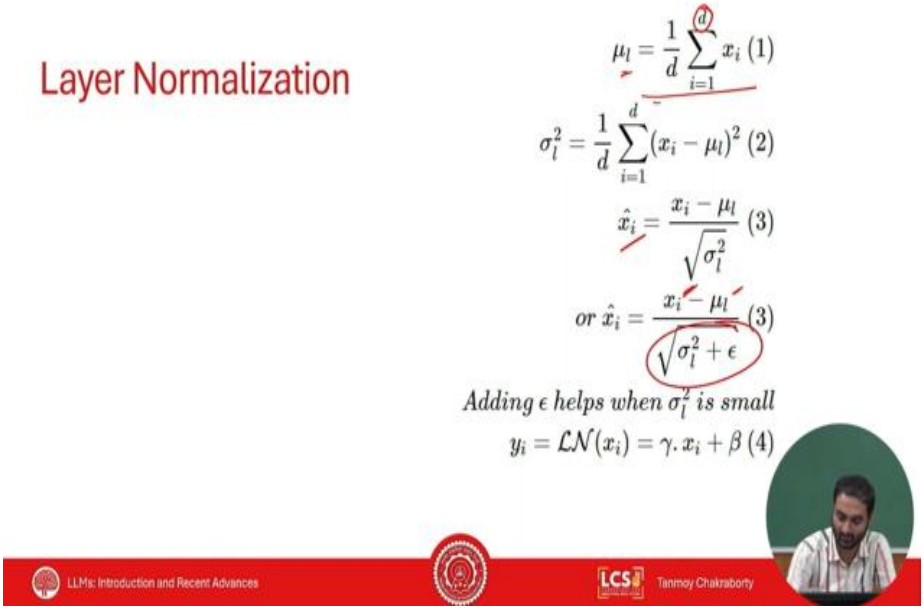
This is not very suitable for a transformer kind of model. Why? Because in a transformer kind of model, which is a sequence model. The sequence length actually varies right. Let's

say you have a batch where there are sequences of length four, there are batches where sequences are of length eight and so on and so forth. So this may not be a suitable option for that kind of cases right, and you easily understand why let's say there's a batch, where there is one sequence of size 4, another sequence of size 8, another sequence of size 16. How do we do this mean? Because in the second instance, you have this fifth element which is not present in the first instance.

So when this varies, this may not be a right approach.



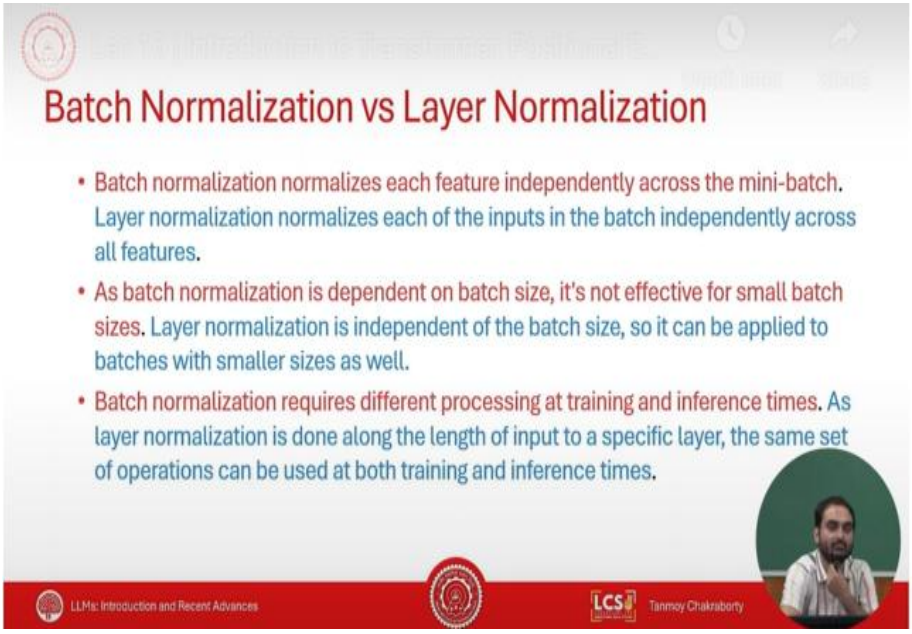




So in layer normalization, what we do? we will not look at all the instances of a batch. There is no concept of batch here at all. We look at every vector and all the elements of this vector will be used to calculate the mean and standard division. Here again, every column indicates an instance right, I will look at column wise entries and then i will basically measure the mean and standard deviation. So one, three, five, seven, mean is four,

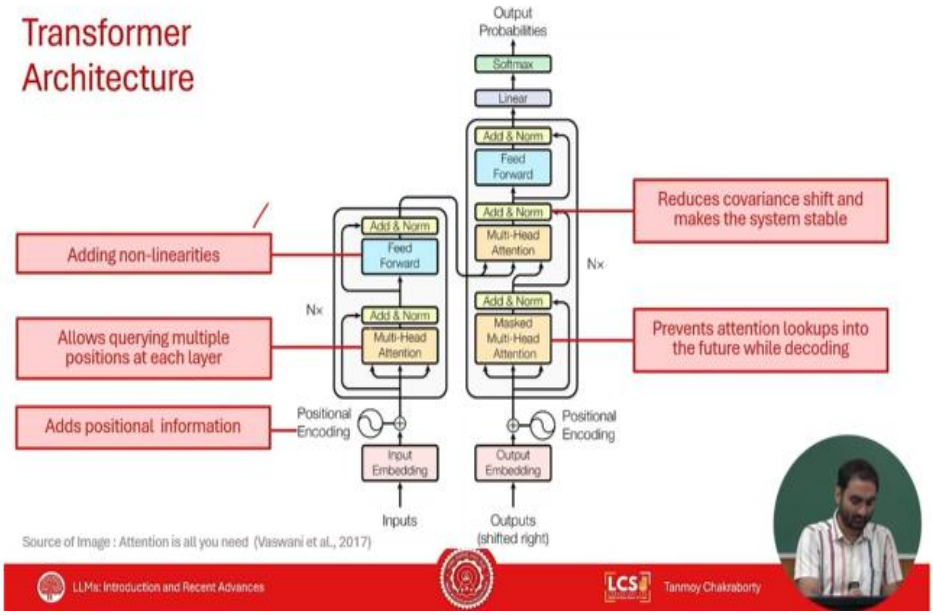
standard deviation is two point two three then i subtract this four right and so on and so forth.

So, this will make sure that each vector follows a distribution. The distribution may vary across vectors, but every vector follows a distribution. Look at here. There are d dimensions for every dimension we basically there are d dimensions and then for every element for every instance I measure the mean and standard deviation then I subtract the mean from the standard deviation and sorry the mean from the element and this is the standard deviation. Here also I will use two learnable parameters if needed okay.



Let's look at the difference between batch and layer normalization. What do you think? Batch I already mentioned that, so in batch normalization, each feature, it normalizes each feature or each element, each entry in the vector independently across the mini batch.

Right and in layer normalization we look at one input and then we look at all the elements of that input and we do layer normalization. So, batch normalization is dependent on the batch size right layer normalization is independent right. But if the batch size is small, it will affect batch normalization, but layer normalization is independent of the size of the batch right does not matter. Batch normalization requires different processing at the training and inference time, now look at this one, right what does it mean? It requires different processing at training and inference time as in layer normalization is done along the length of the input to a specific layer the same set of operations can be used and as both training and inference times.



Okay, so we have actually described all the components of a transformer, right. We have described positional embedding, we have described two ways to do position encoding right the sinusoidal operation and rope right, we described residual connection, we described feed forward networks right which is responsible for doing non-linearity.

Every encoder block consists of mask, self-attention and feed-forward and two add-norm layer and two residual connections and you repeat this block n times. In the decoder part, since this is a decoder block, you do mask attention and then in the cross-attention part here, the query comes from decoder, keys and values will come from the encoder. One point to remember is that it is not the case that the first layer of the decoder will basically attend to the first layer of the encoder in the cross attention formulation. It will not be like that.

Every layer of every decoder layer will attend to the last layer of the encoder block. Okay, so you have n blocks stacked, and the last layer of the encoder block will produce your keys and values. Now every layer of the decoder will attend to those keys and values. Okay, so the this diagram may be a bit wrongly represented, but I hope you understood. Then feed forward network, right? So now I have never mentioned how this model is trained, right? There are so many parameters.

How does this model will be trained? Now, when these decoder blocks are stacked, after the decoder block, let's say you want to perform some task. And in transformer, the paper, the task was machine translation. So next word prediction, given a source sentence, you basically predict the target sentence you generate the target sentence right So what you do? So you have this decoder block right, every token you need to predict right. So you feed start token, it will generate something that will come during the inference time that will come to the next place it will generate something and so on and so forth now during the trading time what is going to happen? During the training time remember transformer will perform all the sequences in parallel process, all the sequence in parallel right. So during the training time you already know the translated version of the input right so all this so let's say this is the last block okay this is the last block of the decoder right Now each of these blocks, each of these heads, each of these tokens will be fed through a linear layer followed by non-linearity short max linear layer followed by non-linearity it will essentially produce a distribution and from that distribution you pick the you see a sample to sample a specific token right.

If it is deterministic you take a gradient you use a greedy approach you take the token which has maximum probability right. You also know what is the ground truth token at that position. In the training set you already have the translated version right so you can basically compute the error, right, some sort of cross-entropy error, right and that gets back propagated. So all this at every at every position of the decoder block you compute the error in parallel. Unlike RNN, where you need to do this sequence, here you are doing in parallel, right? So either you can sum them up and then take the total error and then you backprop.

I think, or let's say you take each of these errors in isolation and then you backprop. So better way is just sum them up and then do the backpropagation. This backpropagation essentially happens in end-to-end fashion. Meaning it will basically come the gradient will flow from here to here get gradient will also flow from here to here right. All the weights of the layers, weights corresponding to the add norm layer, weights corresponding to the feed forward, mask attention, self-attention, and also weights corresponding to the input

embeddings. If you feel that during the, I mean, you need some trainable parameters in the input, but in the position embedding, for example.

I mentioned in the previous part, there are certain cases where we use trainable parameters in the positional embedding. So those parameters will also be updated. Now think the volume of data sets that are needed to essentially train the entire network. We'll talk about training later. This feedforward network will act as a nonlinear module. This will essentially, this self-attention will help you pass messages from one head to another head, right? One position to another position.

I mentioned earlier that the self-attention is nothing but, you know, memory facing operation. And this addNorm, essentially addNorm layer will also help you reduce the covariance shift. What is this covariance shift? This is another important part of the normalization layer. What is covariance shift? When you move from one layer to another layer, it may happen that the entire covariance of the population gets changed because of this operation. So if you do normalization again and again, it will essentially reduce the covariance shift across different layers.

And then you have this mask attention which will help you prevent the attention you know looking up into the future decoding states. This is all about transformer. So, in the next lecture, we will talk about pre-training. We will discuss different types of pre-training techniques, ELMO, BOT and then in the subsequent lectures, we will also talk about GPD and T5 models, encoded only, encoded or decoded only models. Thank you.