**Background and Motivation**

In the word embeddings model, the same token gets the same embedding every time irrespective of the context that they appear in. This is a major pitfall for using embedding in complex tasks. Therefore, we have sophisticated architectures built on embeddings that capture contextual relationships as well.

Attention models transform the default embeddings by analyzing the whole sequence of tokens to make it more representative.

Therefore the goal is to use the word embeddings and transform it so that the word vectors capture and are more representative of the context of the word. This means the same word embedding may get different vectors after transformation.

Now we want input = default embedding and output = embedding which takes into account embeddings of surrounding words as well , i.e, if two embeddings are similar then combine them in some and if they are not similar let them remain the same.

**Attention Architecture (from coursera):**

Encoder-Decoder Architecture sufferers when the input sequence is too long or too short. To deal with long input sequences we use attention models.

Here we have two LSTM/RNN network

1 - Pre-attention network

2 - Post-attention network

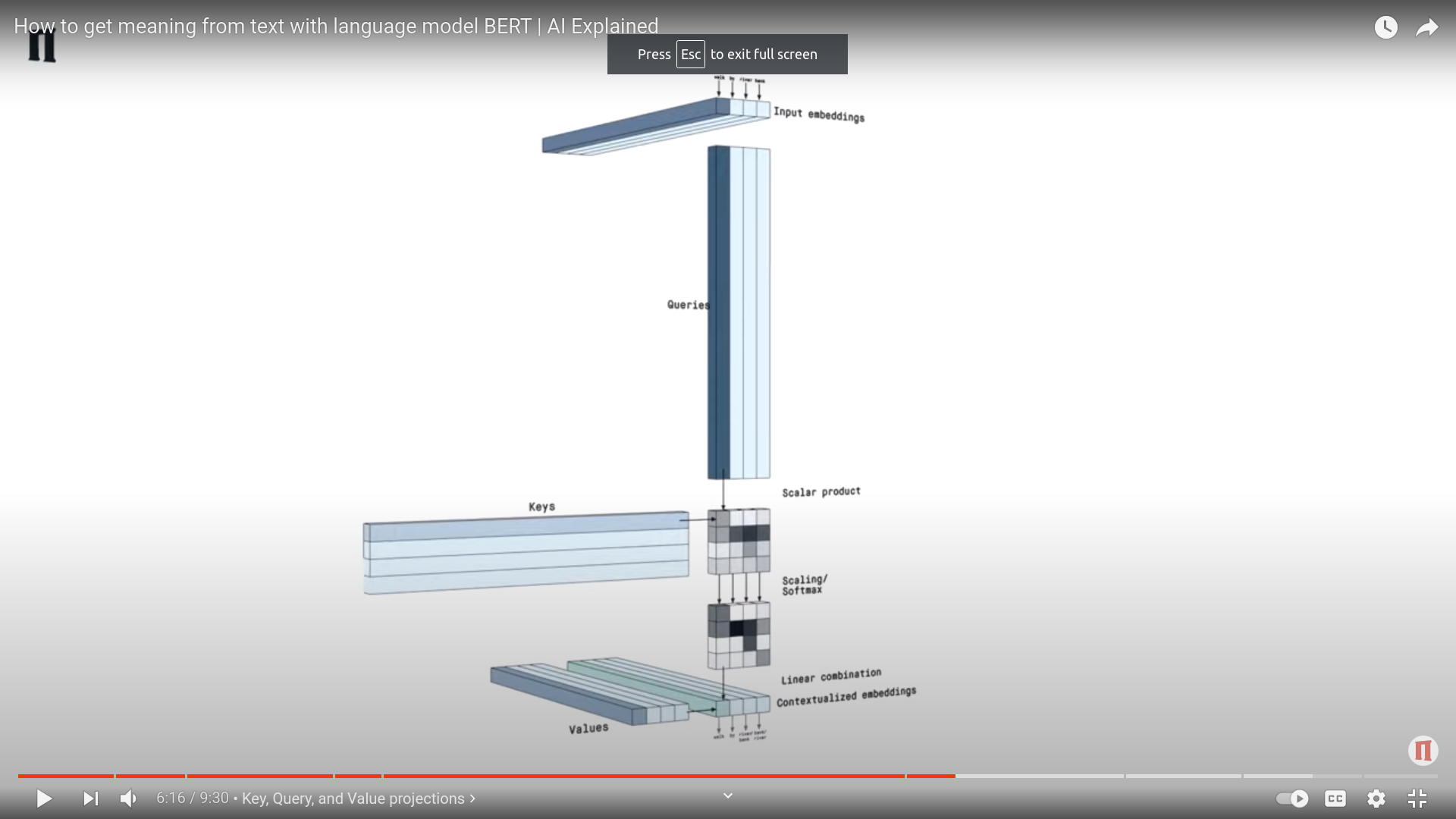
In the pre-attention network we have bidirectional LSTM/RNN that takes input sequences and prev activation across the time steps t. It outputs calculated new activation at each time step.

In the post attention network we have an incoming connection through a block called context block and activation inputs. This are the used to calculate next activations and emits outputs y probabilities at each time step [ labels ]

Now the context block is what connects the pre and post LSTM networks. IN the context box, we have activations output from pre LSTM network and connection signals from every LSTM block in pre LSTMnetwork. The connection strength is a value between 0 and 1 and determines which part to pay attention to and which part to not pay attention to.

Signal strength from all blocks and activation from the current block is multiplied and is what makes the context block. This is then passed as input to the post LSTM architecture. Link to odf here : [Slide29](https://d3c33hcgiwev3.cloudfront.net/CNjR1AdGTs-Y0dQHRs7P6A_f203927163754d95ab2ab917e604b6a1_C5W3.pdf?Expires=1629936000&Signature=csKH82-jngYOw6SSsvrgG~JNH9LApoTZxBscyk-gRCr6ICCB2iZTwGEVU3DbWnadBMFipPoM02AgppkQrAsJh7p1Q8im~5aF7-W0t3mye8numhxC9~XxbtpXhb1oLxrFm4Q8i7yQQLej7ezSw0bD8sueJy4z-3hoOj43WIJVXcI_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

**Self Attention block algorithm (** from [youtube](https://www.youtube.com/watch?v=-9vVhYEXeyQ)**)**



Say we have four words as input = Rainy season is bad.

Step 1 = Take the default embeddings ( word2vec or glove)

Step 2 = Take the dot product of every word with every other word. Dot product gives the strength of relation. So after the dot product we will have a 4x4 matrix where diagonal elements are dot products of a word with itself. In our case the dot product of rainy and season will be strong as they are related. Similarly the dot product of season and rainy will be strong. we don’t project input embeddings as is. Instead we project( decompose) input embeddings into three matrices called queries, keys, values. The dot product that we took here will now be between Queries and keys matrix ( both are projection of input embeddings only)

Step 3 = Now pass this dot product matrix to a softmax activation that will scale the higher value to exponentially higher and lower value to low.

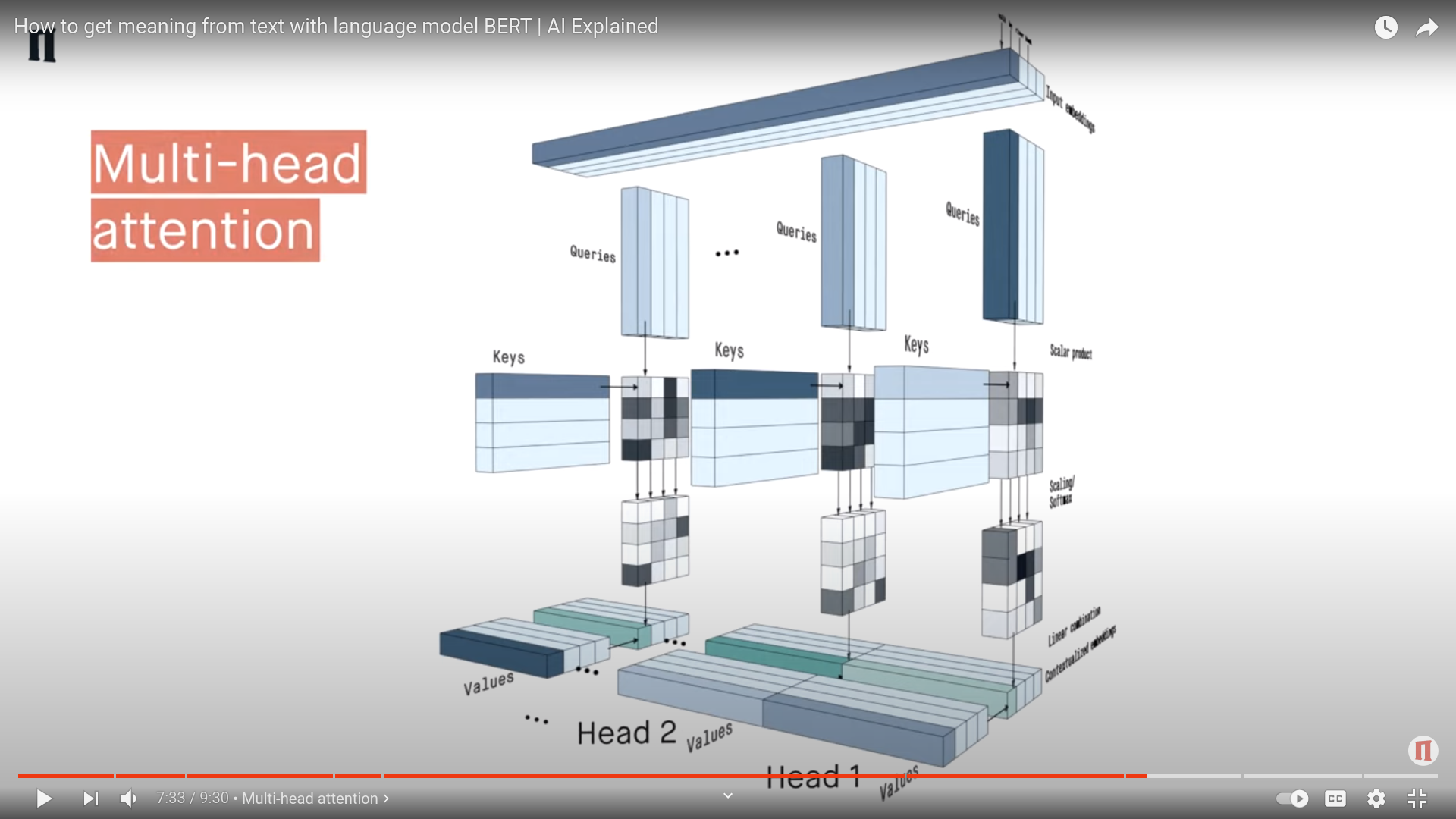
Step 4 = Now take a linear combination of this softmax output and the default input embeddings ( value matrix ). The result is the new embedding which has captured contextual information because this embedding is formed by taking dot product of other tokens with all other tokens. Here, the linear combination is taken with the value matrix of input embedding’s projection.

Therefore, the words with no relation will have the same bedding as default but the ones which have relations like rain, season will have new embeddings which are combinations of both rain, season’s default’s embedding.

In addition to default input embeddings as input we can pass positional embeddings that contain positional information. This enables us to capture both contextual and positional information in the transformed embeddings.

**Multihead attention ( from** [**youtube**](http://youtube.com/watch?v=-9vVhYEXeyQ)**)**

Take the attention block above and repeat it or stack it with different projections of Q,K,V matrices of same input embeddings. This gives us multihead attention



**Encoder decoder architecture (from coursera) :**

Say you are trying to translate Hindi to English text. In encoder decoder architecture we have two RNN networks ( it may also be lstm or gru). First one is called encoder. Just like how CNN encodes images into a dense representation which is then passed to softmax for prediction, here the encoder network takes the input ( sentence) and produces one output which is an encoded representation. We then take this encoded representation and pass it to the second network (RNN/LSTM/GRU). Usually the first activation that is passed ( a\_0) is initialized with zeros but here instead of a\_0 we pass the encoder's output and the decoder network then outputs softmax probabilities at each time step. Greedy approach, that is at each time step choosing T’th highest value of y’s from all y’s softmax probabilities, is not a good idea. We instead then do beam search to choose that y out of all y’s probabilities that maximizes the joint probability of all y’s across time step.

For example : input : hum idar jaa rahe hai

In our encoder architecture, each of the four words acts as an input at each time step t =1 to 4 and then at the end output y is given which is an encoded presentation.

This encoded representation is then passed to the decoder network where at each time step t= 1=4 we output English vocab probabilities. We choose the one that maximizes P(y1/x1)\* P(y2/y1,x2) \*P(y3/y2,x3) \* ….. By using beam search.

Link to pdf : [here](https://d3c33hcgiwev3.cloudfront.net/CNjR1AdGTs-Y0dQHRs7P6A_f203927163754d95ab2ab917e604b6a1_C5W3.pdf?Expires=1629936000&Signature=csKH82-jngYOw6SSsvrgG~JNH9LApoTZxBscyk-gRCr6ICCB2iZTwGEVU3DbWnadBMFipPoM02AgppkQrAsJh7p1Q8im~5aF7-W0t3mye8numhxC9~XxbtpXhb1oLxrFm4Q8i7yQQLej7ezSw0bD8sueJy4z-3hoOj43WIJVXcI_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

**Transformers**

Transformers have two blocks :

Encoders :

Step 1 : Take default word embeddings and positional vectors as input.

Step 2 : Dot product of word embeddings with positional vector

Step 3 : pass the resulting word embeddings as input to multihead block which gives Query, Keys, Values as output ( contextual word embeddings projections)

Step 4 : Norm and residual connection

Step 5 : Pass the resulting output to feedforward for further processing ( what’s the output of this ?)

Step 6: repeat this 3 - 5 for N times resulting in N blocks inside encoder

Decoders :

Step 1: Initialize input as start token and pass it to first masked multihead after doing all the default embeddings

Step 2 : The first multi head here is masked meaning it doesn't look ahead in future

Step 3: Now this output acts as value input at the next multihead. Tw other incoming inputs here are quey and key from the encoder blocks output

Step 4 : output ne Q,K,V pairs and pass it to feed forward layers

Step 5 : Repeat this for N blocks

Step 6 : At the last layer, pass the output to classifier which will then pass it to softmax outputting class probabilities ( say for instance, vocabulary)

Step 7 : Pass the result back to decoder’s first masked multihead block and repeat

Note : initially the word is only start token in decoder then start token is passed to decoder block which outputs word w1, then w1 is passed back as input to decoder which is again masked and then sent ahead which outputs w2 which is again passed as input and this goes on till end token is generated.

**BERT**

Bert is transformer but with only stacked encoders network