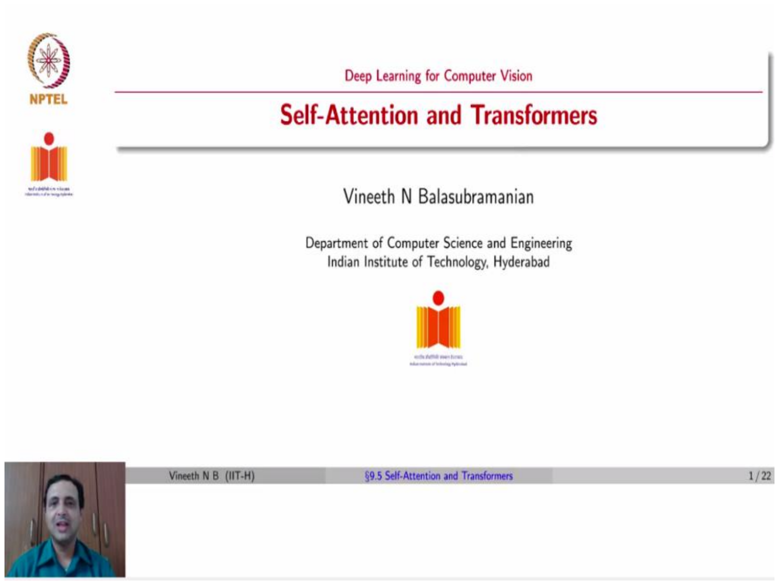
Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 60
Self-Attention and Transformers

(Refer Slide Time: 00:14)



Deep Learning for Computer Vision

Self-Attention and Transformers

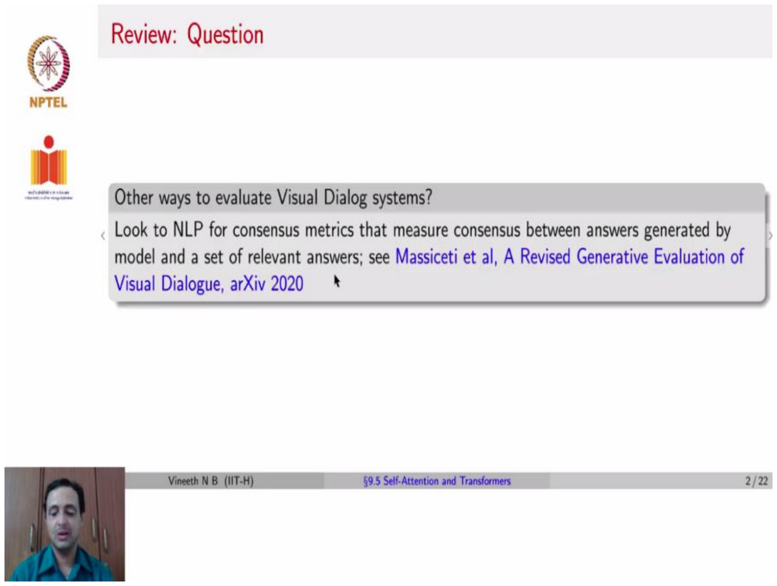
Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 1 / 22

For the last lecture of this week, we will look at a topic that is becoming extremely popular in recent months, which is Self Attention and Transformers.

(Refer Slide Time: 00:30)



Review: Question

Other ways to evaluate Visual Dialog systems?

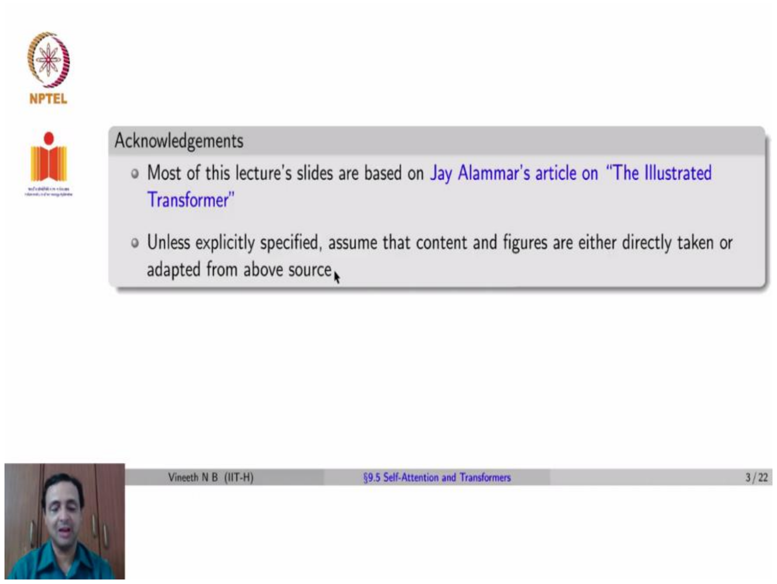
Look to NLP for consensus metrics that measure consensus between answers generated by model and a set of relevant answers; see Massiceti et al, A Revised Generative Evaluation of Visual Dialogue, arXiv 2020

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 2 / 22

One question that we left behind in an earlier class on Visual Dialogue was, what are other ways in which one can evaluate visual dialogue systems? Hope you had a chance to think about it or look for it online. One answer to this question is to look to natural language processing for getting performance metrics from the text domain, which could measure consensus between answers generated by a model and a relevant set of answers.

If you would like to know more about such an approach, please see this recent work called a Revised Generative Evaluation of Visual Dialogue. This work also talks about how when one can generate a set of relevant answers.

(Refer Slide Time: 01:31)



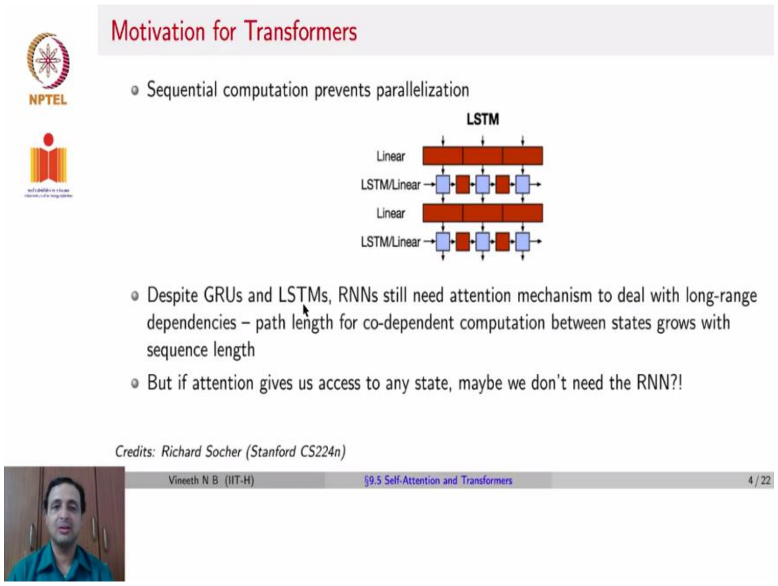
Acknowledgements

- Most of this lecture's slides are based on Jay Alammar's article on "The Illustrated Transformer"
- Unless explicitly specified, assume that content and figures are either directly taken or adapted from above source

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 3 / 22

So let us now come to Transformers and Acknowledgement. Most of this lecture slides are based on Jay Alammasr's excellent article on The Illustrated Transformer. So you can assume that any images that are not credited are adapted or borrowed from Jay Alammasr's article.

(Refer Slide Time: 01:55)



Motivation for Transformers

- Sequential computation prevents parallelization

- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long-range dependencies – path length for co-dependent computation between states grows with sequence length
- But if attention gives us access to any state, maybe we don't need the RNN?!

Credits: Richard Socher (Stanford CS224n)

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 4 / 22

Let us first try to understand why does one wants to go beyond RNNs or LSTM's for sequence modeling. So sequential computation prevents parallelization. So if you had a series of layers, let us assume you have a linear layer, followed by an LSTM layer, followed by a linear layer, followed by an LSTM layer. When we say LSTM layer, we mean a series of LSTM blocks that process a sequence of inputs.

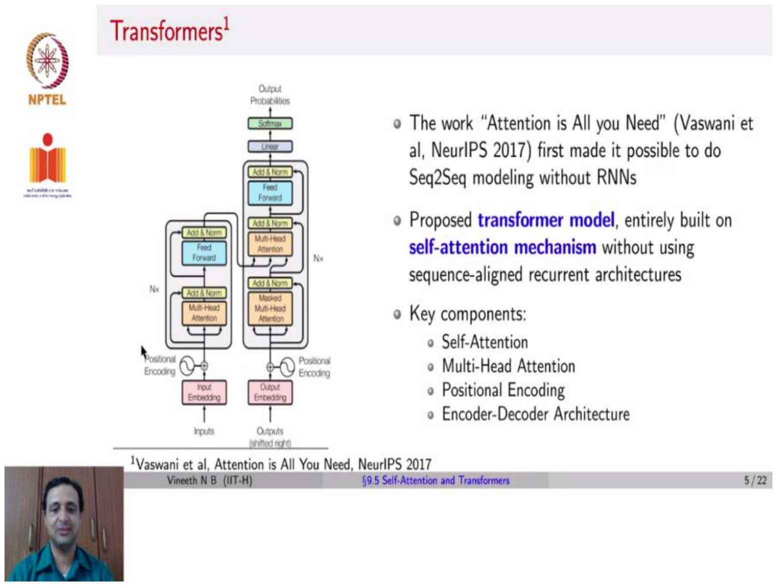
So if you had an architecture that is similar to this, you could consider some of the architectures that we had for complex tasks such as VQA or Visual Dialogue could have had such a sequence of layers. When you have an architecture, such as this, it is not possible to parallelize the processing of information through such a pipeline. Can we overcome this is a question we want to try to answer?

Further despite the availability of GRUs and LSTMs, RNNs can only maintain information for a limited period. If you have very long time-series data, for example, a chapter of a book, will an RNN hold the information or a GRU or an LSTM hold information for a very long time series? Possibly not, even in that scenario, one perhaps needs attention to focus on the right part of the previous content.

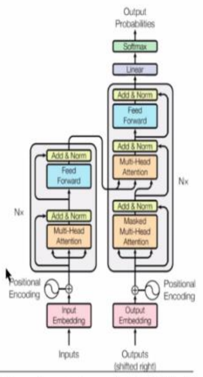
We saw this with models for saying visual question answering or visual dialog, all of which used attention, not all of which at least some of which used attention to be able to focus on the relevant parts of the previous sequence information. So if we are anyway using attention in RNNs, why use RNNs at all? Attention can anyway tell us which part of a sequence to look at while giving a specific output.

So if you are doing, say, machine translation, while you are generating a particular word, you can decide which part of the input sequence should I focus on. You maybe do not need an entire RNN to hold the hidden state across multiple time steps. That is the hypothesis for Transformers, which we will talk about in this lecture.

(Refer Slide Time: 04:52)



Transformers[1]



- The work "Attention is All you Need" (Vaswani et al, NeurIPS 2017) first made it possible to do Seq2Seq modeling without RNNs
- Proposed **transformer model**, entirely built on **self-attention mechanism** without using sequence-aligned recurrent architectures
- Key components:
  - Self-Attention
  - Multi-Head Attention
  - Positional Encoding
  - Encoder-Decoder Architecture

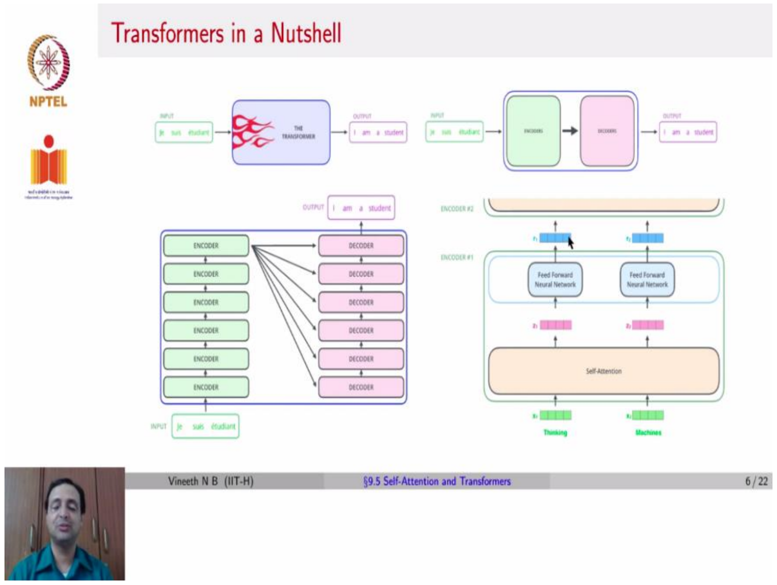
[1]Vaswani et al, Attention is All You Need, NeurIPS 2017

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 5 / 22

Transformers were first introduced in this work called Attention is all you need. Published in NeurIPS of 2017, which introduced an encoder-decoder framework to perform sequence to sequence modeling without any RNNs. This work proposed the transformer model, which introduced the Self Attention mechanism without any recurrent architectures. There were few key components of this architecture.

A concept is known as Self-attention, then multi-head attention, positional encoding, and overall encoder-decoder architecture. We will see each one of these in detail. What you see on the left here is the overall architecture. Hopefully, this would become clear over the next few slides.

(Refer Slide Time: 05:57)

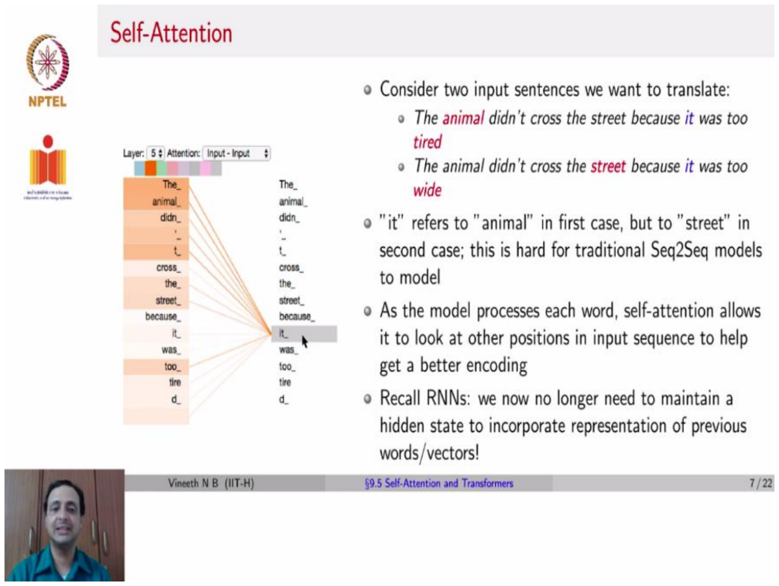


So a quick summary of how Transformers work. If you considered a machine translation task, let us say you want to translate this input into the output with states I am a student, we are proposing the use of an encoder-decoder architecture where in the encoder, you have several encoder modules, each of which feeds into the encoder at the next level. And all of these, and the encoder at the highest level feeds into each of the decoders, which has similar staggered architecture.

And the final output is I am a student, each encoder has several components inside it, there is a self-attention module, which outputs a vector for every input word. So if you had a sequence of inputs, for each element of that sequence, you get a certain representation. As an output of the self-attention module, this representation is passed through a feed-forward neural network to give you another vector is output, which is passed on to the next encoder layer, and so on.

Let us see each of this competence in detail.

(Refer Slide Time: 07:32)



Self-Attention

- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models to model
- As the model processes each word, self-attention allows it to look at other positions in input sequence to help get a better encoding
- Recall RNNs: we now no longer need to maintain a hidden state to incorporate representation of previous words/vectors!

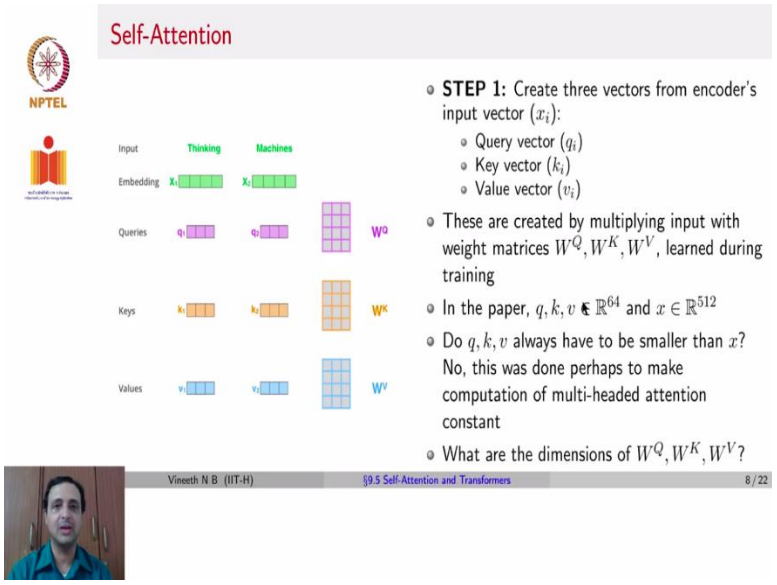
Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 7 / 22

Let us start with self-attention, which is one of the unique features that this work introduced. So let us consider two input sentences that we would like to translate. One sentence reads, the animal did notxthe street, because it was too tired. And the second sentence is the animal did notxthe street because it was too wide. For a human, it is evident that the first it corresponded to the animal.

And the second corresponded to the street. But for an RN based more on an RNN based model, this is difficult to catch and that is the reason self-attention brings value. Self-attention, allows the model to look at what are the other words that need to be paid attention to, while you process one particular word. The reason it is called self-attention is given a sequence when you are processing one element of that sequence, you make a copy of that sequence and see which other parts of the same sequence are important to process this element.

And that is why it is called self-attention. So now if you had to draw a parallel to RNNs, you would see that you perhaps do not need a hidden state to maintain the history of previous sequence inputs. Whenever you process one element of a sequence, at that time, you process an attention vector over an entire sequence. You do not need to store a compressed hidden state that has all the sequence information in one vector or one matrix. That again is the idea of transformers.

(Refer Slide Time: 09:50)



Self-Attention

- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training
- In the paper, $q, k, v \in \mathbb{R}^{64}$ and $x \in \mathbb{R}^{512}$
- Do q, k, v always have to be smaller than x? No, this was done perhaps to make computation of multi-headed attention constant
- What are the dimensions of W^Q, W^K, W^V?

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 8 / 22

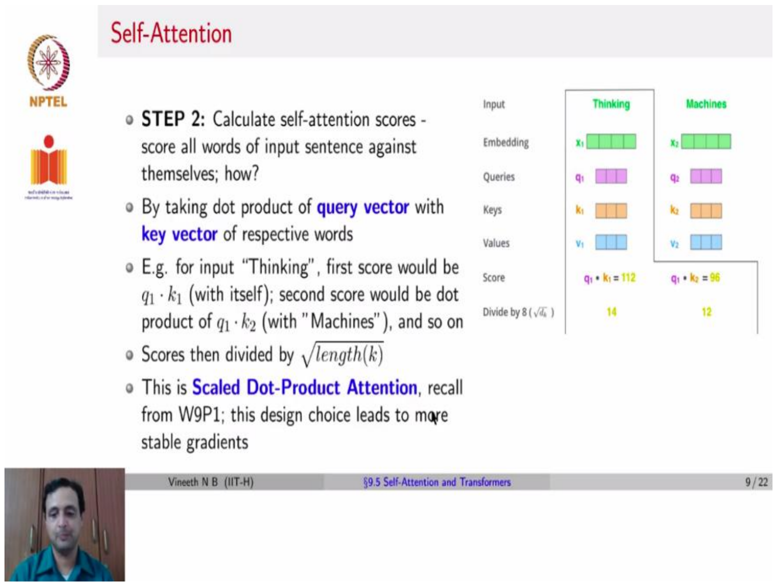
So let us see how self-attention is implemented. The first step is given an input, input vector, which is an embedding. So let us say you have a phrase thinking machines. So, you consider the first element of that sequence thinking, you would get a word embedding. Once again, assume that this is done using some text processing techniques to go from thinking to embedding, an embedding is a vector representation of the word thinking.

Similarly, X2 here is the vector representation of the word machines. So, given these inputs to the self-attention module, there are three vectors that the self-attention module creates these vectors are known as the Query vector q, a key vector k, and a value vector v. We will see how these are used very soon. But how are these vectors created? To create the q, k, and v vector, one uses learnable weight matrices WQ, WK, and WV which are multiplied by the input X to give us q, k, and v.

So in this particular work q, k and v, were 64-dimensional vectors, and X by itself, which is the embedding of each word in the sequence was a 512-dimensional vector. One question here, does q k v always have to be smaller than X? In this case, you see that X is 512 dimensional? q, k, and v are 64 dimensional? Does this always have to be the case for Transformers to work? The answer is No.

This was perhaps done in this work to make the computation of multi-headed attention constant, we will see this very soon. What are the dimensions of W^Q, W^K, W^V, you should be able to get that from X and q k v, you would have 512x64. Those are the weights that are learned while you train the transformer end to end.

(Refer Slide Time: 12:28)



Self-Attention

- **STEP 2:** Calculate self-attention scores - score all words of input sentence against themselves; how?
- By taking dot product of **query vector** with **key vector** of respective words
- E.g. for input "Thinking", first score would be $q_1 \cdot k_1$ (with itself); second score would be dot product of $q_1 \cdot k_2$ (with "Machines"), and so on
- Scores then divided by $\sqrt{length(k)}$
- This is **Scaled Dot-Product Attention**, recall from W9P1; this design choice leads to more stable gradients

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 9 / 22

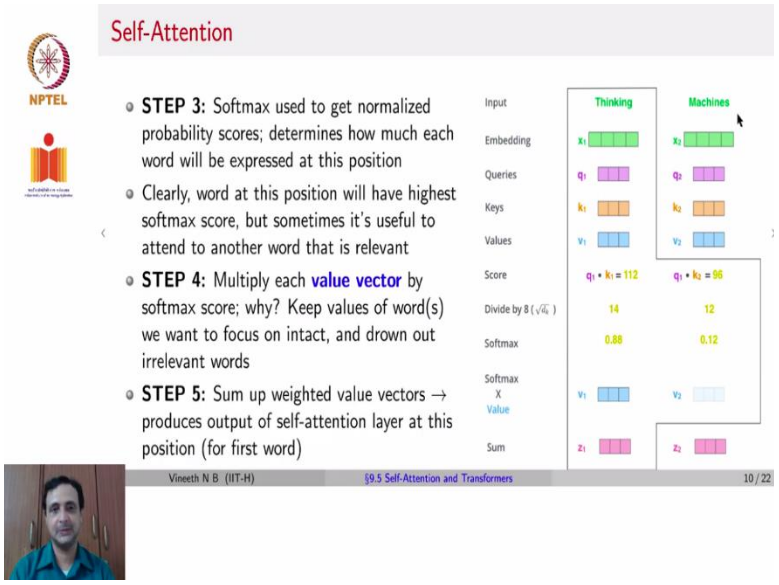
So once you get your q, k, and v vectors for each word in your input sequence, step two in self-attention is to compute what is known as self-attention scores, which is the score of all words of the input sentence against this particular word under consideration. So if you had the word thinking, which is what you are processing, you would compare this with every other word in the sequence and compute a self-attention score. How is that computed?

Let us see, it is by taking the dot product of the query vector with the key vector of the respective words. So for this input, thinking, the first score would be $q_1 \cdot k_1$, that is the query vector, which is q corresponding to the word thinking, and the key vector corresponding to the word thinking $q_1 \cdot k_1$. Similarly, you would compute $q_1 \cdot k_2$ for the second word in the sequence, $q_1 \cdot k_3$ for the third word in the sequence, and so on.

These scores are then divided by the root of the length of k, which is the dimension of the query, key, and value vector. In our case, the dimension was 64. So the root of the length would be 8. So the score $q_1 \cdot k_1$, which was 112 would be divided by eight and you get 14. Similarly, $q_1 \cdot k_2$ was 96. And that would get divided by 8 and you get a 12. This is known as scale dot product attention.

You can recall our discussion in the first lecture of this week, where we spoke about several kinds of attention, alignments, and scaled dot product attention, where you take a dot product and then scale the value is one such alignment mechanism. Why do we do this here, scale dot product, attention helps get more stable gradients by scaling the values to a manageable range, instead of letting it go too high, or too low, and the variance is too much.

(Refer Slide Time: 15:12)



Self-Attention

- **STEP 3:** Softmax used to get normalized probability scores; determines how much each word will be expressed at this position
- Clearly, word at this position will have highest softmax score, but sometimes it's useful to attend to another word that is relevant
- **STEP 4:** Multiply each **value vector** by softmax score; why? Keep values of word(s) we want to focus on intact, and drown out irrelevant words
- **STEP 5:** Sum up weighted value vectors → produces output of self-attention layer at this position (for first word)

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 10 / 22

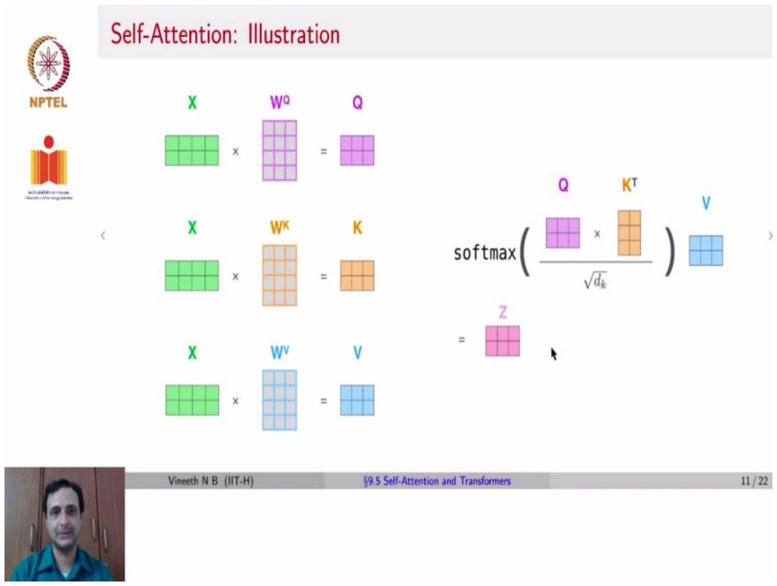
Once this is done, the next step is to use softmax to get a distribution of the attention of each of the words in the sequence with respect to the word under consideration. So once you get the scores 14 and 12, you can do a softmax on them to get a weight for the first word or wait for the second word with respect to the first word, we are still focusing on the still processing the first word here.

Here, this score of the first word with respect to the first word is likely to have the highest weight because you would expect that to process the first word, the first word is the most important. However, we still have weight or attention vectors attention values, over other words in the sequence, which also add value to processing the current word. As the next step, we multiply the softmax output with the value vector that we got earlier.

Remember, we had a query key and a value vector, we now use that value vector with the softmax output to now get a weighted representation of the value vector. Similarly, you will have a weighted representation of the second value vector, and so on, for every sequence of every input in the sequence. And we finally then do a weighted sum of all of this v1, v2, and so on vectors to get a final z1, z2, so on and so forth.

So z1 here is the weighted sum of v1 into softmax plus v2 into its softmax value, plus v3 into its softmax value will together form z1. Similarly, z2 would be formed when we process the word machines.

(Refer Slide Time: 17:29)



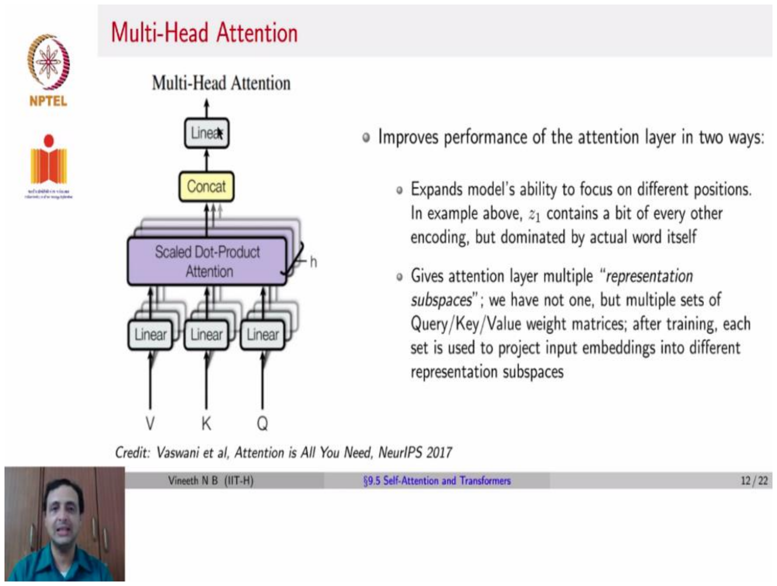
Self-Attention: Illustration

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 11 / 22

What do we do with this z1s? Before we go there, let us try to summarize our discussion so far. So you have your input embedding x, you see two rows in x here, corresponding to vector representations for two words, thinking, and machines in this particular case. Then you have the matrices of W^Q, W^K, W^V those are learnable weights. And when you multiply x and that weight matrix, you get a Q, K, and V.

Then you do a dot product between Q and K, scale it by the root of the dimension of the K vector, do a softmax on it, multiply with the v vector, and that gives you your z vector for each of your words in the input sequence.

(Refer Slide Time: 18:24)



Multi-Head Attention

- Improves performance of the attention layer in two ways:
  - Expands model's ability to focus on different positions. In example above, z_1 contains a bit of every other encoding, but dominated by actual word itself
  - Gives attention layer multiple "*representation subspaces*"; we have not one, but multiple sets of Query/Key/Value weight matrices; after training, each set is used to project input embeddings into different representation subspaces

Credit: *Vaswani et al, Attention is All You Need, NeurIPS 2017*

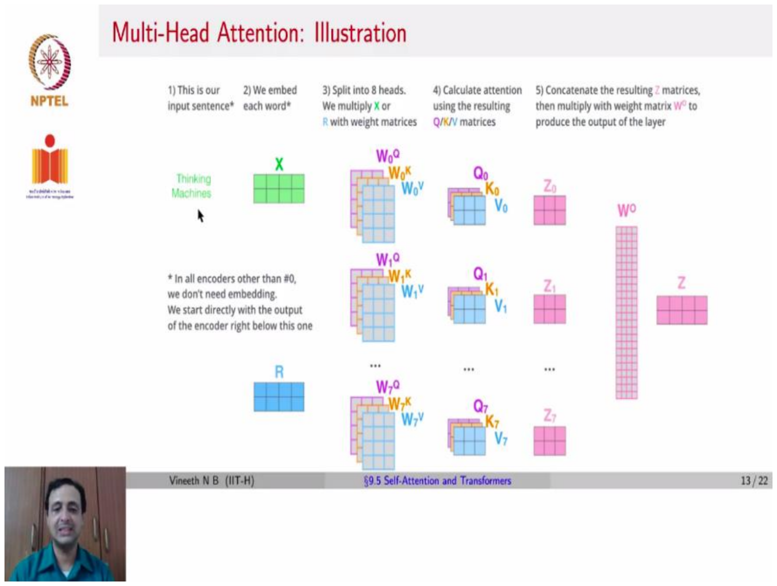
Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 12 / 22

Now let us move on to Multi-Head Attention, we have seen self-attention so far. Next, the next we will talk about the component known as multi-head attention multi-head attention are these different layers that you see in the illustration here, instead of using only one W^Q, or one W^V or one W^K, transformers suggests having multiple such of W^Q, W^K, W^Vs so you would get multiple such query vectors, key vectors, and value vectors.

So and you would do multiple such self-attention computations. And you finally then concatenate and send them through a linear layer. How does this help expand the model's ability to focus on different positions? So, different W^Q, W^K, W^Vs weight matrices could help the model focus on different parts of the sentence, which could together enrich the processing of the current word.

Secondly, it also gives our self-attention layer a sense of multiple representation spaces, instead of choosing only one set of weight vectors and getting these different representation spaces and then concatenating them and sending through a linear layer gives us better performance at the end. This part of Transformers is known as Multi-Head Attention.

(Refer Slide Time: 20:11)



Multi-Head Attention: Illustration

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

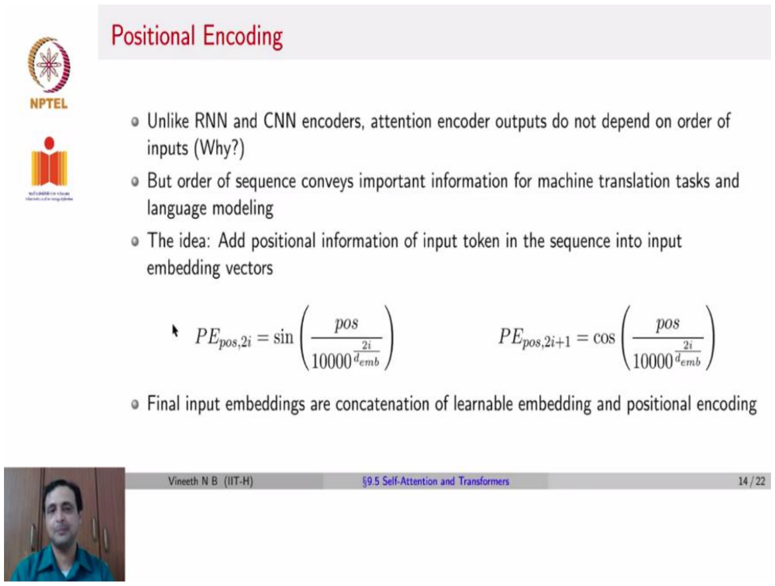
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 13 / 22

Here is the illustration, so if you have your input sentence, in our case, it is thinking machines, we embed each word. Each row of x is an embedding of thinking and machines. We have multiple W^Q, W^K, W^Vs. In this case, we see eight such heads, resulting in eight query vectors, key vectors, and value vectors, we get eight different Zs, and then we use a linear layer to combine all of those Zs to get one Z for each input of the sequence.

So, you see here that there is something called R, R simply indicates that in your transformer architecture, there are multiple encoder layers. The first encoder layer receives x as the input. Every other encoder layer receives the output of the previous encoder layer as input that is denoted by R. So the original sentence, and the words may not be given as input to the second encoder in that set of stacked encoders. But it could be the output of the first encoder that is fed there.

(Refer Slide Time: 21:36)



Positional Encoding

- Unlike RNN and CNN encoders, attention encoder outputs do not depend on order of inputs (Why?)
- But order of sequence conveys important information for machine translation tasks and language modeling
- The idea: Add positional information of input token in the sequence into input embedding vectors

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{emb}}}}\right) \qquad PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{emb}}}}\right)$$

- Final input embeddings are concatenation of learnable embedding and positional encoding

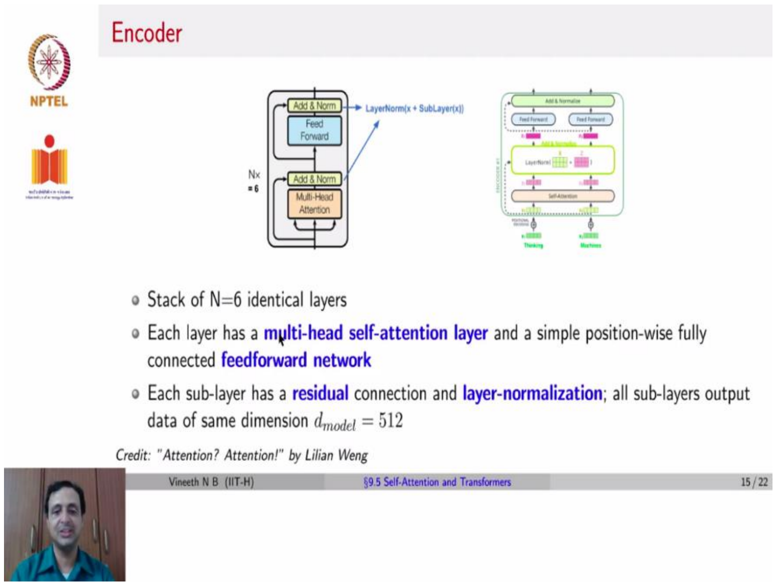
Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 14 / 22

A third component of the Transformers architecture is known as Positional Encoding. Unlike CNN and RNN encoders, Attention encoded outputs do not depend on the order of inputs. Why? Because you can choose to focus on what you want, irrespective of the order of the inputs. Having said that, it still is important as to which position a particular word is in a sentence to make meaning out of it.

So how do we bring that value into an attention model into a Self-attention model? That is done in Transformers using an approach known as positional encoding. So the position of a word in a sequence is given as a separate input, along with the x embedding. How is this position encoded? You could do it in different ways but the paper suggests one specific way, although this may not be the only way it takes the position of that particular word with respect to the sequence and processes it as a sin and cosine signal.

And that becomes a positional encoding of that word in the sequence. This is given as input along with what we saw as x in the previous slide. So given thinking machines, if you took the word thinking, you would get a certain word embedding, you concatenate the position encoding along with that, that becomes your input to yourself attention layer. So the role of position encoding is to bring some value of where in the sequence, a specific input which you are currently processing is.

(Refer Slide Time: 23:41)



Encoder

- Stack of N=6 identical layers
- Each layer has a **multi-head self-attention layer** and a simple position-wise fully connected **feedforward network**
- Each sub-layer has a **residual** connection and **layer-normalization**; all sub-layers output data of same dimension $d_{model} = 512$

Credit: "Attention? Attention!" by Lilian Weng

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 15 / 22

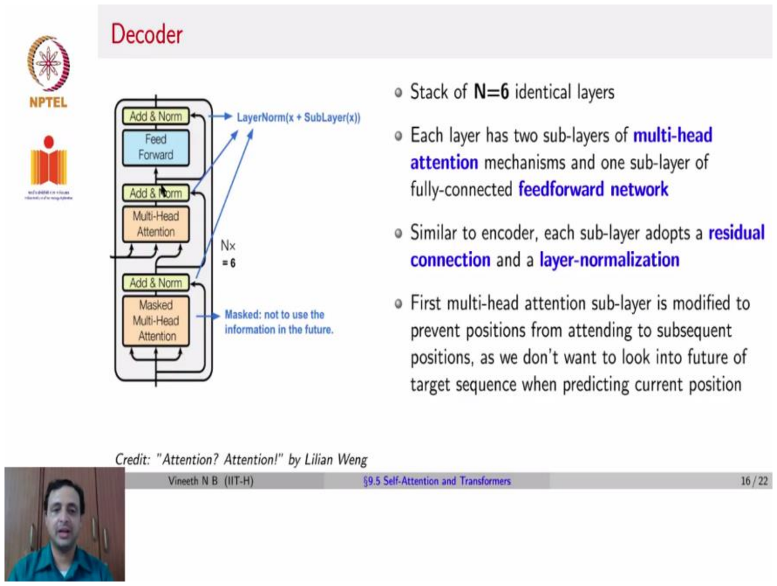
Finally, to come to the architecture of the entire model. As we mentioned, so far, it is an encoder-decoder architecture. Let us first see the encoder. The encoder is a stack of six identical layers in this particular transformer that was proposed. Each of such six layers looks somewhat like this, what you see in the image here. So you have a multi-head self-attention layer, which we have already seen self-attention, take multiple heads, concatenate and then have a linear layer to get a vector input.

And that is fed into a fully connected network feed-forward network that you see on top, but between the multi-head attention and the feed-forward layer, there is a residual connection. And there is also a normalization that happens before that output is fed to the feed-forward layer. So you first perform multi-head attention. Then the input also comes through a skip connection directly to the output of multi-head attention, where you add and normalize.

The Z output that you get out of multi-head attention, and the $X + P$ input, that is X plus Positional encoding input that you get before the multi-head attention, you would add them, normalize them, that is fed into a feed-forward network. Once again adding and normalizing and is the output of one encoder module. You would have six, six such encoder modules in this transformer architecture.

And all the sub-layers that you see within this particular encoder layer, output data of the same dimension 512 in this work.

(Refer Slide Time: 25:45)



Decoder

- Stack of N=6 identical layers
- Each layer has two sub-layers of **multi-head attention** mechanisms and one sub-layer of fully-connected **feedforward network**
- Similar to encoder, each sub-layer adopts a **residual connection** and a **layer-normalization**
- First multi-head attention sub-layer is modified to prevent positions from attending to subsequent positions, as we don't want to look into future of target sequence when predicting current position

Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
LayerNorm(x + SubLayer(x))
Nx = 6
Masked: not to use the information in the future.

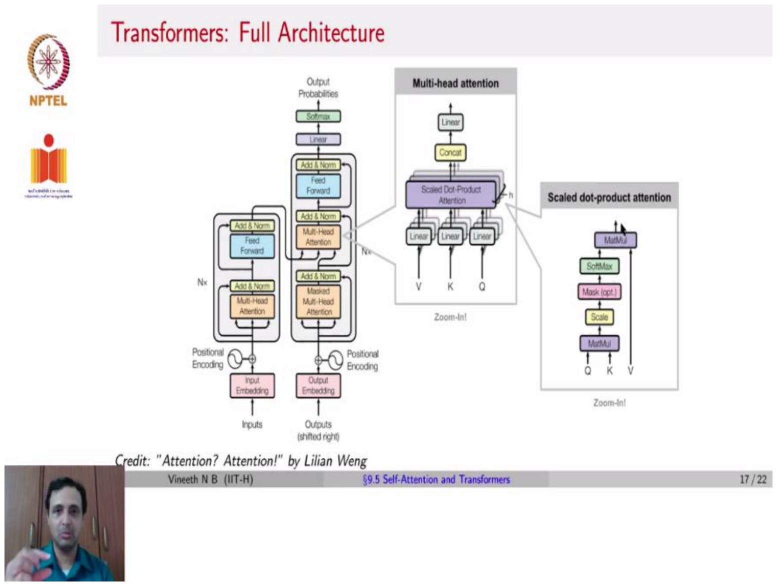
Credit: "Attention? Attention!" by Lilian Weng

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 16 / 22

Similarly, the decoder also had six identical layers. Each layer again has multi-head attention and a feed-forward network. Again, you have a residual connection and a normalization step that is done before the output of multi-head attention is given to the feed-forward layer. There is only one difference here, this initial step also performs what is known as masked multi-head attention. The goal here is to not let the architecture see further words in the output sequence when it is processing one specific word.

So if you are translating from English to Hindi, you would not want if you are now trying to process the third word in the output, at that time, you can use attention on the first two words, but you do not want to use attention on the remaining words, because the model is not supposed to know them while predicting the third word. This is done by masking the attention mechanism in the first decoder and then the outputs are passed on as it is to further multi-head attention mechanisms.

(Refer Slide Time: 27:10)



Transformers: Full Architecture

Credit: "Attention? Attention!" by Lilian Weng

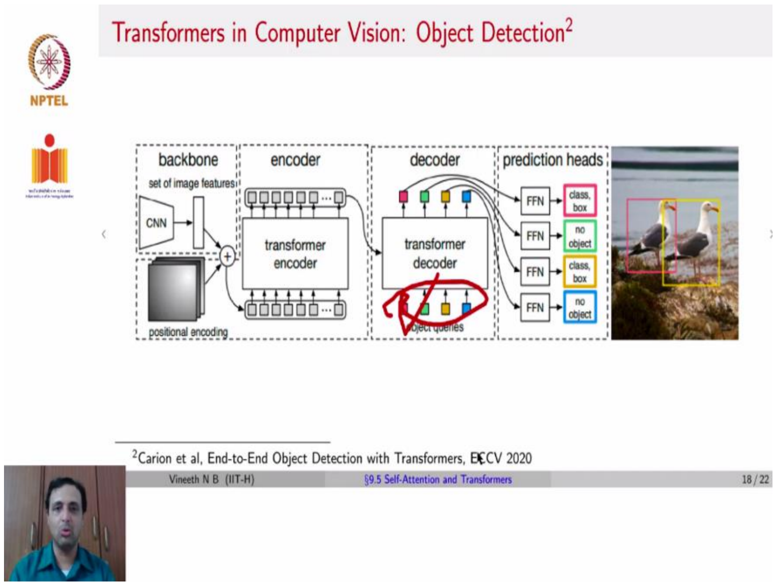
Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 17 / 22

That is the architecture of the decoder. So, here is a revisit of the complete architecture again the input, input embedding, then the positional encoding concatenated, you have the multi-head attention mechanism, then you have a skip connection, add a normalize, feed-forward add and normalize, you have ideally six such stacked encoders and similarly, you have the decoder as we just spoke, which has a masked multi-head attention mechanism.

Add a normalize than a multi-head attention mechanism add a normalize, feed-forward add a normalize and you once again have six such modules stacked on top of one another. So, the scaled dot product attention can also be visualized this way, where you have Q K and V, you first perform a matrix multiplication of Q and K that gives you your dot product, then you scale the dot product, then you may choose to mask it optionally depending on which module you are talking about.

Masking is done only in this first part of the decoder here, then you apply the softmax then multiplied with V and that becomes the output of the scale dot product attention module.

(Refer Slide Time: 28:37)



Transformers in Computer Vision: Object Detection[2]

[2]Carion et al, End-to-End Object Detection with Transformers, ECCV 2020

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 18 / 22

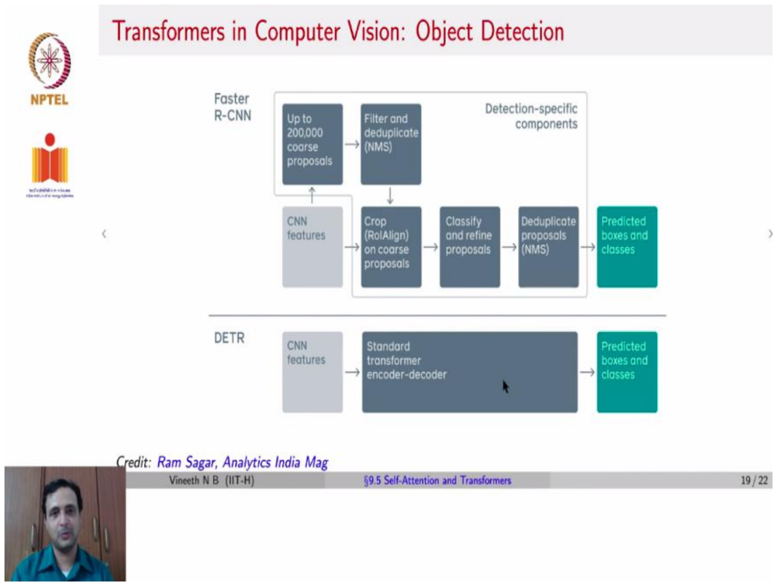
Now, that was about transformers, for General Text processing or Sequence Processing problems. Coming back to computer vision, how can Transformers be used in Computer Vision. So this is being explored over the last few months very recent work performed one published in ECCV 2020 on Transformers for object detection, were given a CNN which is broken down into a set of image features.

You also have a positional encoding of where each feature belongs to the input image. These are concatenated and given to a transformer encoder, which processes each patch those outputs are given to a transformer decoder. And then you have you give object queries here, object queries are taking a patch from the input and then trying to find out what is the object inside each of those patches.

The transformer decoder processes those object queries and gets an output that is passed to a feed-forward network, which finally predicts each of those object queries. Object queries are certain patches of the input image, which are queried for what object is there in that specific patch. So as you can see in the outputs here, you could have a class label and a bounding box offset.

Or you could simply say no object, which means that patch corresponds to the background. You see here similarly, for other regions, you have a class label with a bounding box offset, or a no object. So this was the use of Transformers for Object Detection, published a couple of months ago at ECCV.

(Refer Slide Time: 30:37)



Transformers in Computer Vision: Object Detection

Faster R-CNN
Up to 200,000 coarse proposals
Filter and deduplicate (NMS)
Detection-specific components
CNN features
Crop (RoIAlign) on coarse proposals
Classify and refine proposals
Deduplicate proposals (NMS)
Predicted boxes and classes
DETR
CNN features
Standard transformer encoder-decoder
Predicted boxes and classes

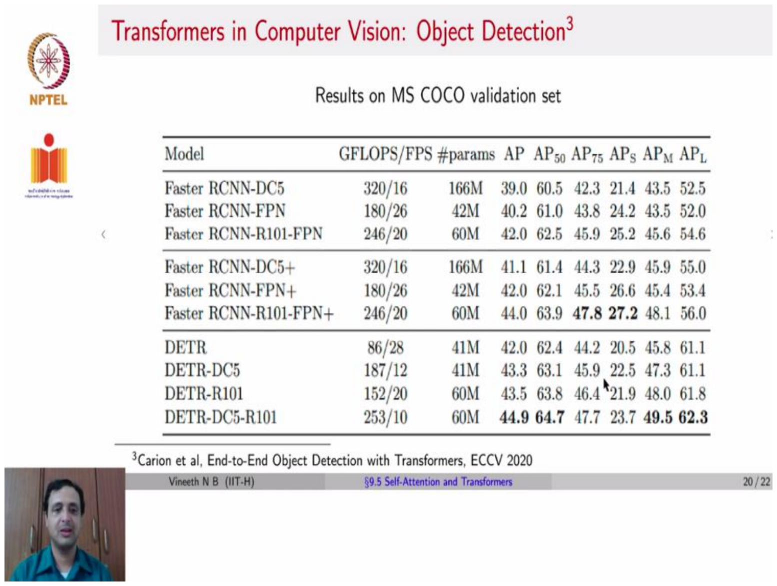
Credit: Ram Sagar, Analytics India Mag

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 19 / 22

And here is a visualization of how this approach simplifies the object detection pipeline. So if you consider a faster R CNN, if you recall, you have your initial CNN features from which you extract perhaps about 200,000-course proposals, you filter de-duplicate, do non-maximum suppression, and then do an ROI align of the proposals to let them a map, map to specific regions in the input image. And then you finally predict your bounding boxes and classes.

While using Transformers here replaces all of these steps with a single end-to-end trainable encoder-decoder.

(Refer Slide Time: 31:28)



Transformers in Computer Vision: Object Detection[3]

Results on MS COCO validation set

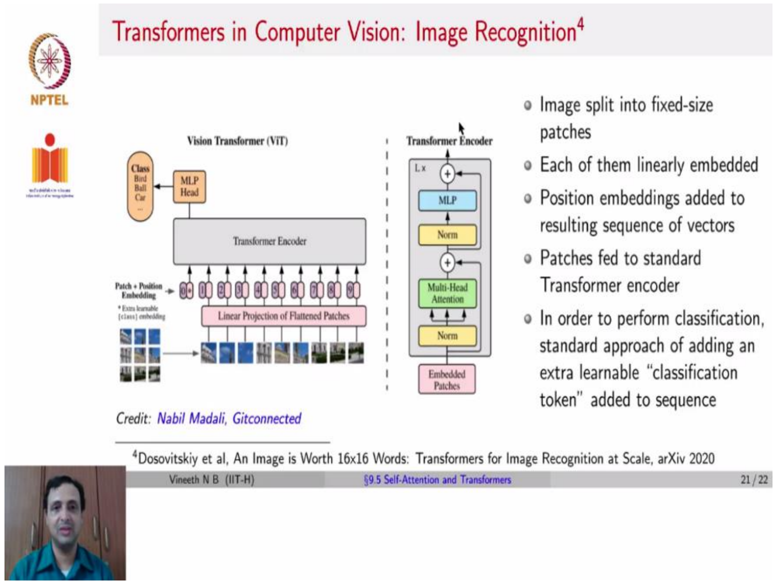
Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

[3]Carion et al, End-to-End Object Detection with Transformers, ECCV 2020

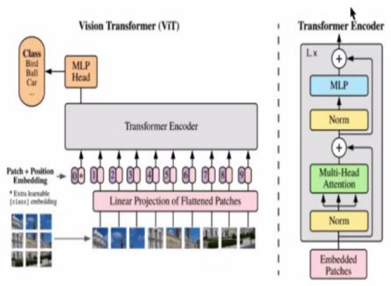
Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 20 / 22

This work showed that they could beat the performance of faster R-CNN methods on the MS COCO validation set, you can see the numbers here of average precision, AP stands for average precision, AP at 50 stands for average precision at a particular threshold. And you can see that the transformer models seem to outperform state-of-the-art faster R-CNN models across a range of different metrics.

(Refer Slide Time: 32:01)



Transformers in Computer Vision: Image Recognition[4]



Credit: Nabil Madali, Gitconnected

- Image split into fixed-size patches
- Each of them linearly embedded
- Position embeddings added to resulting sequence of vectors
- Patches fed to standard Transformer encoder
- In order to perform classification, standard approach of adding an extra learnable "classification token" added to sequence

[4]Dosovitskiy et al, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, arXiv 2020

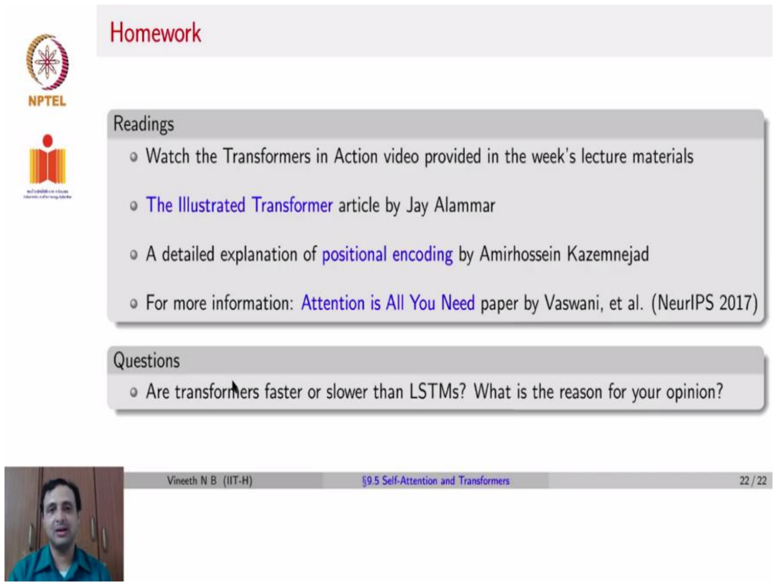
Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 21 / 22

More recently, just about a month back, there was a work published in the archive on Transformers for image recognition. This approach takes a set of fixed-size patches from the image. Each of those is linearly projected to get a vector representation. Positional encodings of each of these patches with respect to the original image are also added to form the inputs. These inputs are fed to a standard transformer encoder. And finally, there is a feed-forward network or an MLP head that is used to get the final classification.

To perform the classification this approach also uses an extra learnable classification token here an embedding for the class label, which is also given as input to a transformer encoder. The transformer by itself is composed of a first normalization step, then multi-headed attention, then a skip connection to add the input back to the normalization of feed-forward network or MLP. And then concatenation with the output of multi-head attention and then the output of the encoder.

This is very recent work, and the space of Self Attention and Transformers are completely revolutionizing how we look at Deep Learning methods for Sequences, and probably now even images.

(Refer Slide Time: 33:43)



Homework

Readings
- Watch the Transformers in Action video provided in the week's lecture materials
- The Illustrated Transformer article by Jay Alammar
- A detailed explanation of positional encoding by Amirhossein Kazemnejad
- For more information: Attention is All You Need paper by Vaswani, et al. (NeurIPS 2017)

Questions
- Are transformers faster or slower than LSTMs? What is the reason for your opinion?

Vineeth N B (IIT-H) §9.5 Self-Attention and Transformers 22 / 22

The homework there is a Transformers in action video provided along with your lecture slides for this week, please watch it to understand how Transformers work. Also the excellent article by Jay Alammar on the illustrated transformer. And if you need to understand positional encoding better the specific article by positional encoding on positional encoding. And finally, if you need more information, the original paper attention is all you need.

Let us end this with one question, are Transformers faster or slower than LSTMs? What do you think? And why do you think so? Think about it and we will discuss this the next time.