Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

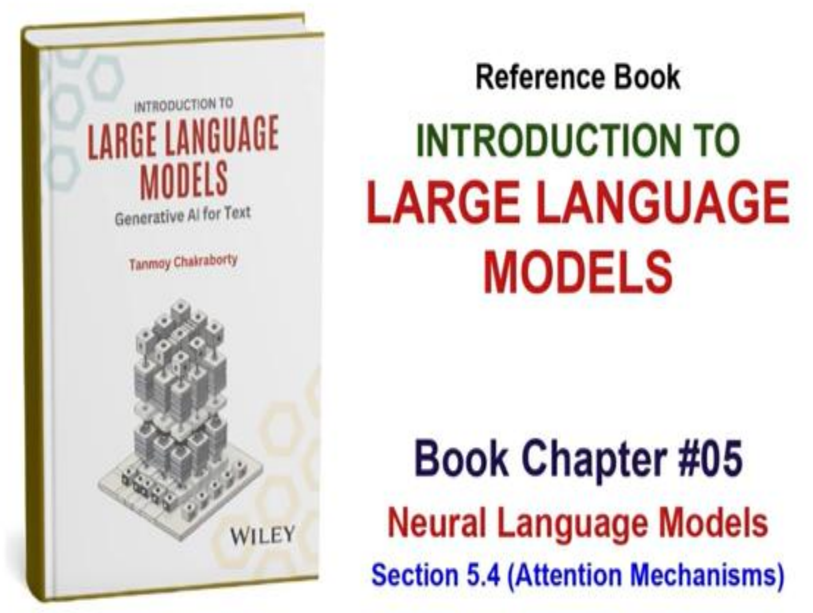
Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

Lecture 14

Attention in Sequence-to-Sequence Models

Hello everyone, in today's class we will start attention.



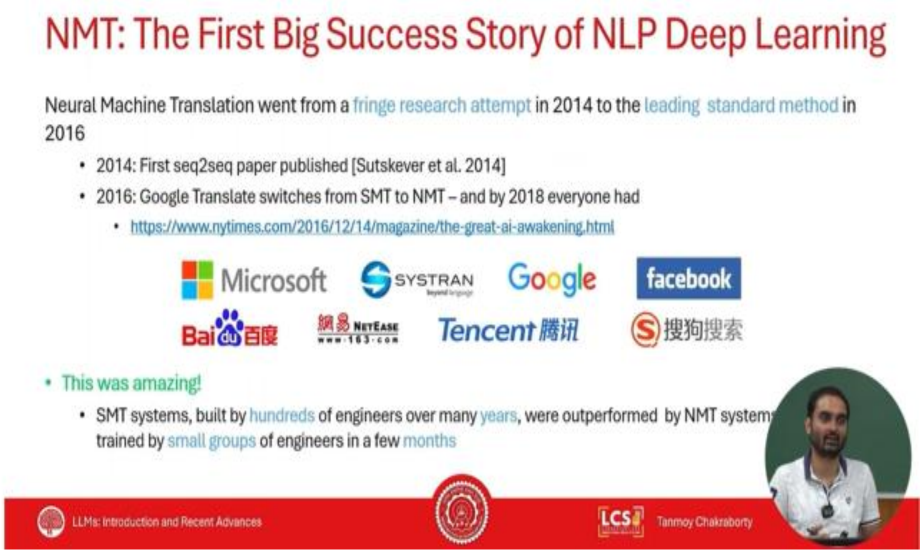
Reference Book

INTRODUCTION TO LARGE LANGUAGE MODELS

Book Chapter #05

Neural Language Models

Section 5.4 (Attention Mechanisms)



NMT: The First Big Success Story of NLP Deep Learning

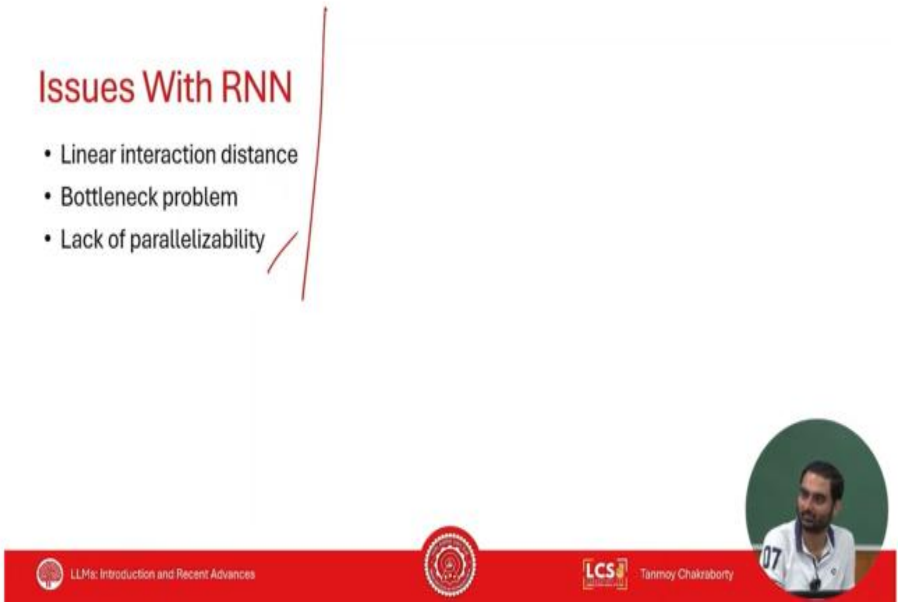
Neural Machine Translation went from a fringe research attempt in 2014 to the leading standard method in 2016

- 2014: First seq2seq paper published [Sutskever et al. 2014]
- 2016: Google Translate switches from SMT to NMT – and by 2018 everyone had
  - https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

Microsoft
SYSTRAN
Google
facebook
Baidu百度
網易 NetEase
Tencent 腾讯
搜狗搜索

- This was amazing!
  - SMT systems, built by hundreds of engineers over many years, were outperformed by NMT systems trained by small groups of engineers in a few months

So sequence to sequence problem was very popular in 2014, in fact it turned out to be one of the popular models which basically you know outperformed all the statistical machine translation models that were out there right, 40 years of statistical machine learning history, statistical machine translation history. Right suddenly the sequence to sequence model outperformed all the statistical models okay and then in 2016 right people noticed that you know some more tweaks or some more modifications can also be done on top of sequence to sequence okay in 2016 a new method was proposed which is called the attention



Issues With RNN

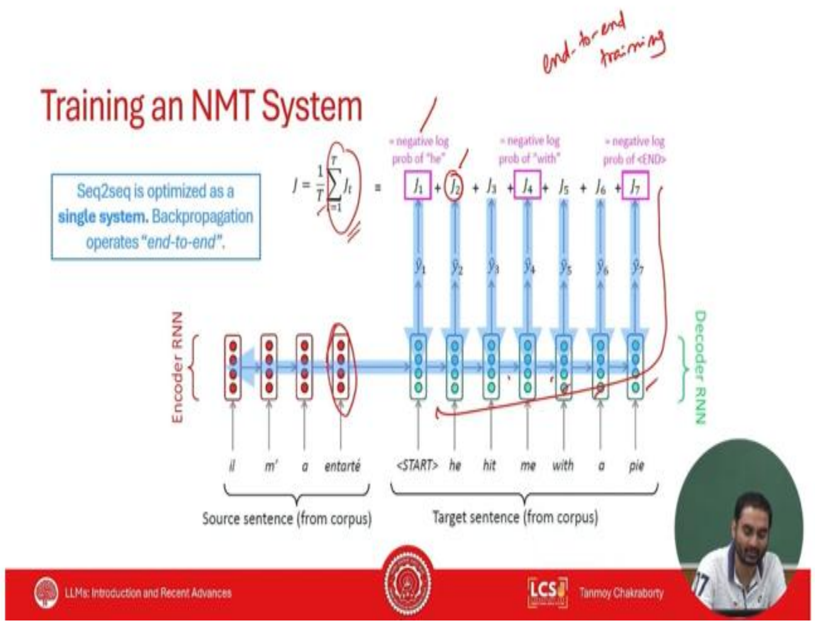
- Linear interaction distance
- Bottleneck problem
- Lack of parallelizability

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

Okay now let's look at attention here So now so far sequential sequence is also RNN. There are two RNNs right. There are three problems which still persist. What are the three problems? The first problem is you can't access all the tokens in parallel right, the first problem.

The second problem if you think about it very carefully the sequence to sequence model essentially considers each pair of interactions. Let us say hidden state t, hidden state t plus 1, this interaction, hidden state 1, hidden state 2, hidden state t plus 5, hidden state t plus 6, every consecutive pair of interactions are considered in a similar way. So the interaction distance is considered as linear, right. So the way we treat 1 versus 2, the way we treat i-1 t, we will treat the same way, let us say t plus 5, t plus 6.

The third problem is very important.

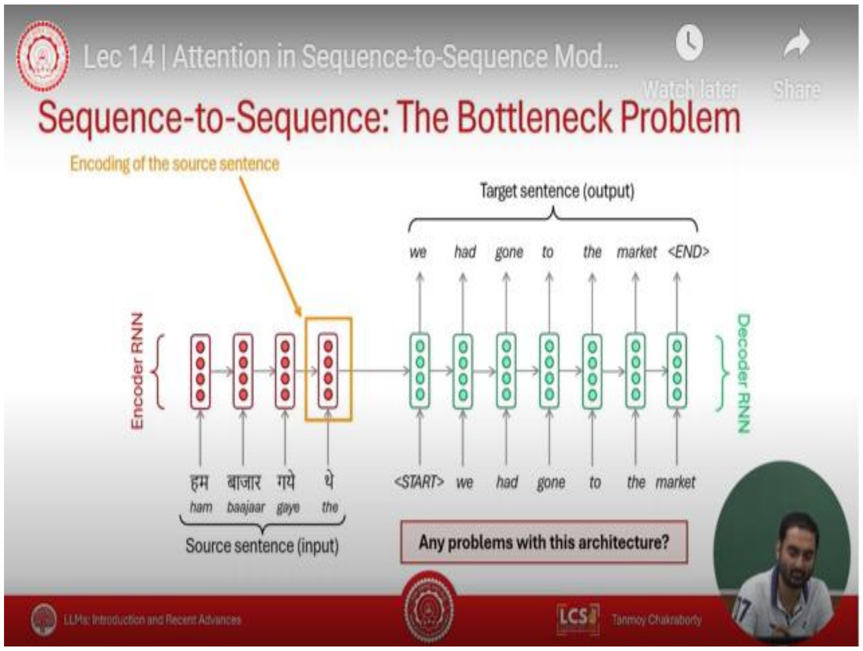


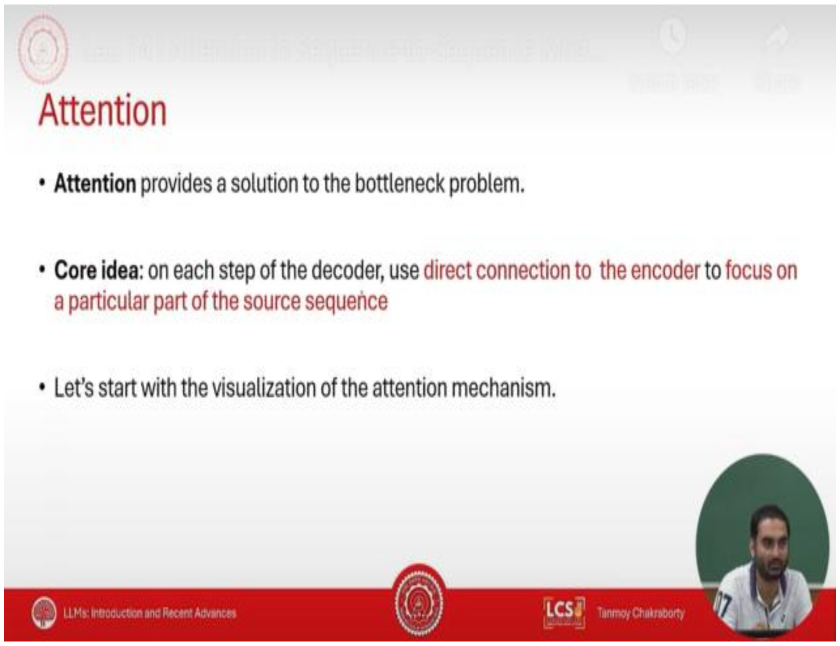
end-to-end training
Training an NMT System
Seq2seq is optimized as a single system. Backpropagation operates "end-to-end".
$J = \frac{1}{T}\sum_{t=1}^{T} J_t = J_1 + J_2 + J_3 + J_4 + J_5 + J_6 + J_7$
= negative log prob of "he"
= negative log prob of "with"
= negative log prob of <END>
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$
Encoder RNN
Decoder RNN
il m' a entarté
<START> he hit me with a pie
Source sentence (from corpus)
Target sentence (from corpus)
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

The third problem is called the bottleneck problem. Look at this one, okay. So this guy, this is the final hidden state of the decoder. Right so this input is fed to the, this is the final, instead of the encoder this input will be fed to the decoder right, the decoder has access to only this hidden state right, for the decoder this is the input, decoder doesn't have access to these hidden states.

So, this guy has a lot of responsibility of remembering the entire input sequence. So, there is a bottleneck on this in terms of remembering the entire sequence of the input. If you are not able to compute it properly, you will not be able to process it. Think about it very carefully. The word he corresponds to the word il.

The word pi corresponds to the word this, right? But at this time, at this stage, this hidden state has no access to the first hidden state of the encoder. So due to vanishing gradient problem, due to other problems, over smoothing and et-cetera, it will happen that this hidden state will essentially forget about the first hidden state, right? And when you come here, this guy will completely forget about this one. So instead of this, if we had a direct connection from decoder to encoder, that could have solved this problem maybe. Let's say there is a direct connection from he to il, a direct connection from pi to this one. Right then that could have solved the problem because through the shortcuts right information can move from encoder to decoder.


Lec 14 | Attention in Sequence-to-Sequence Mod...
Watch later
Share
Sequence-to-Sequence: The Bottleneck Problem
Encoding of the source sentence
Target sentence (output)
we had gone to the market <END>
Encoder RNN
Decoder RNN
हम बाजार गये थे
ham baajaar gaye the
<START> we had gone to the market
Source sentence (input)
Any problems with this architecture?
LLMs: Introduction and Recent Advances
LCS2
Tanmoy Chakraborty

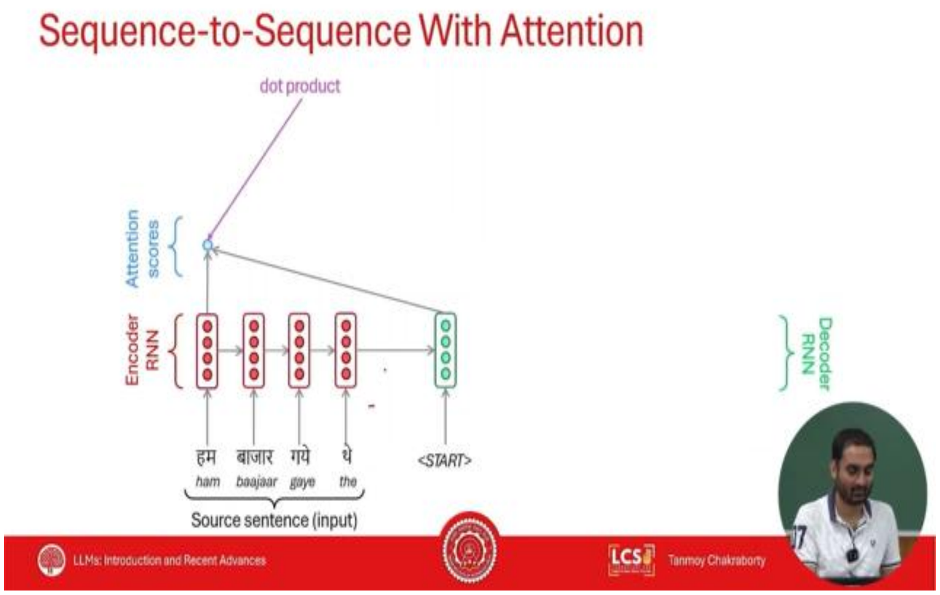


Attention

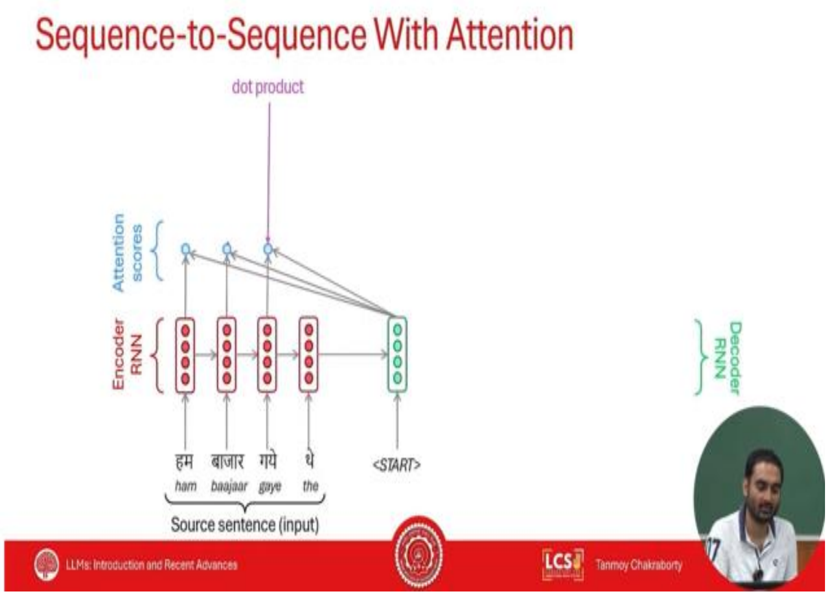
- **Attention** provides a solution to the bottleneck problem.
- **Core idea**: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence
- Let's start with the visualization of the attention mechanism.

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

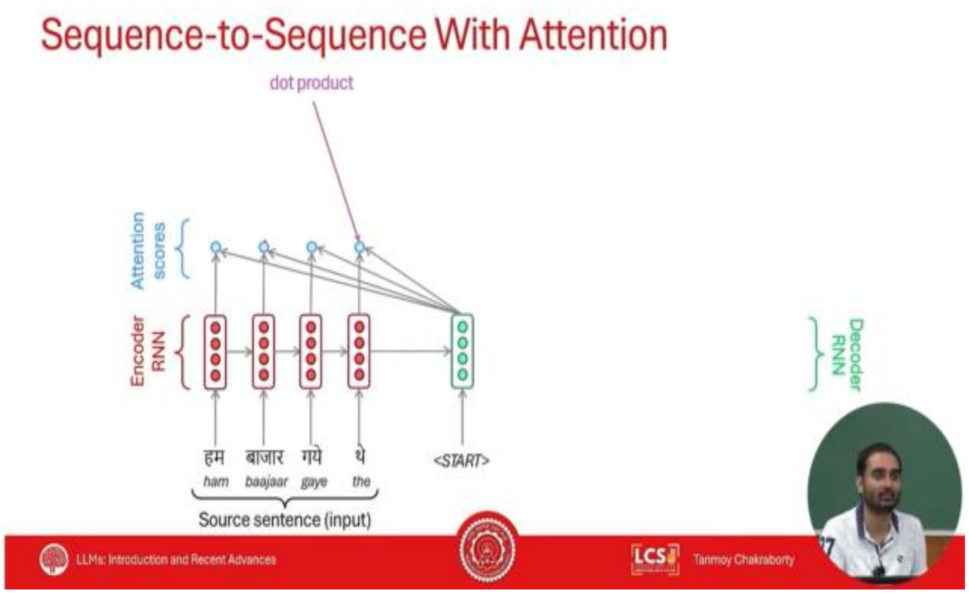
So this is the philosophy of attention I already mentioned about this bottleneck problem right So what is attention? The core idea here is that on each step of the decoder We use a direct connection to the encoder to focus on a particular part of the source sequence.



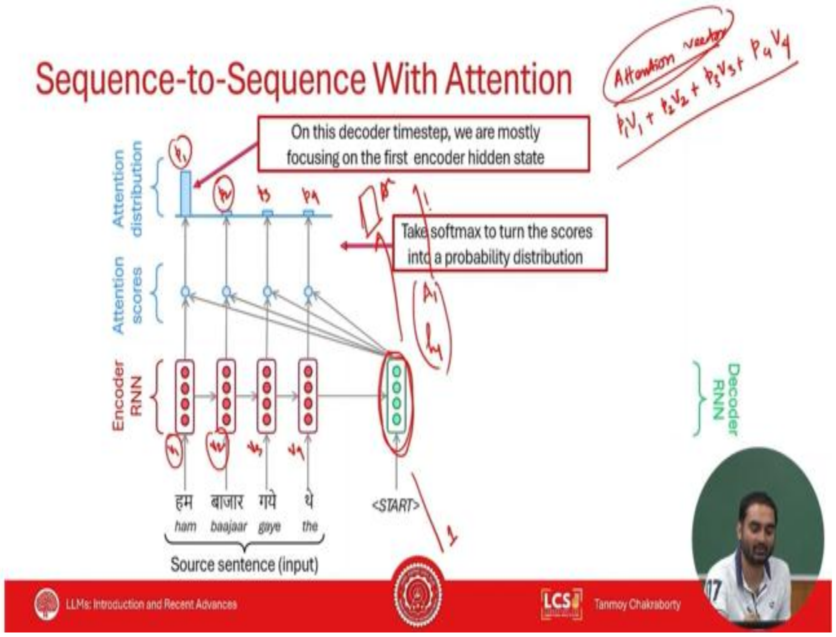
Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention

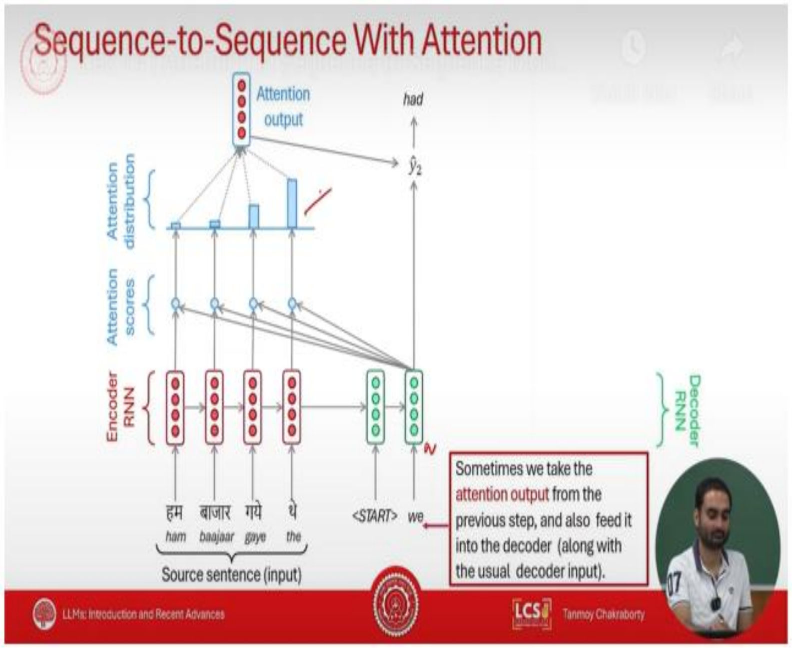


Sequence-to-Sequence With Attention

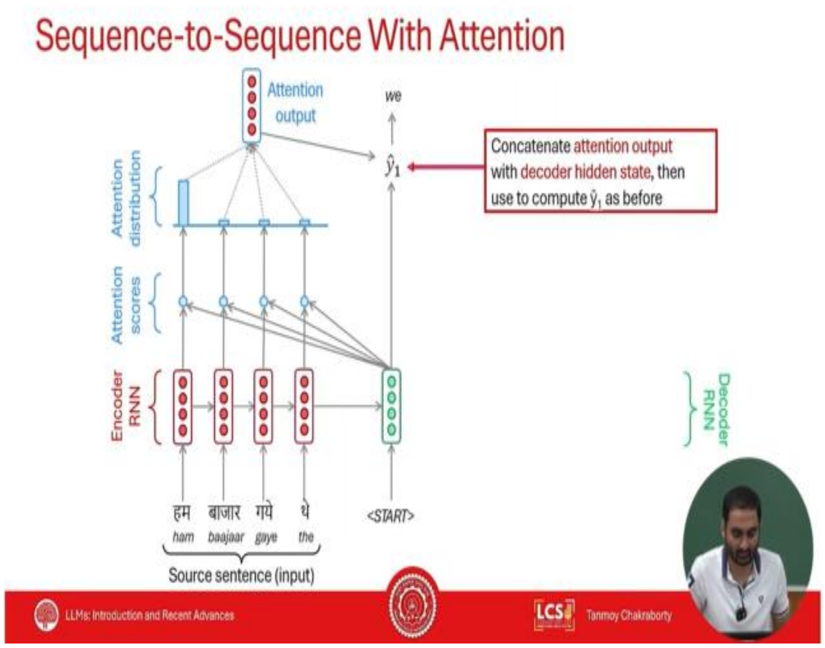


Sequence-to-Sequence With Attention

Attention vector
$p_1V_1 + p_2V_2 + p_3V_3 + p_4V_4$
On this decoder timestep, we are mostly focusing on the first encoder hidden state
Take softmax to turn the scores into a probability distribution
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
p_1 p_2 p_3 p_4
v_1 v_2 v_3 v_4
हम बाजार गये थे
ham baajaar gaye the
<START>
Source sentence (input)
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty



Sequence-to-Sequence With Attention



Sequence-to-Sequence With Attention

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

So let's look at the architecture first and then we will understand. Let's take a Desi example. Ham baajaar gaye the. And you want to convert this Hindi to English.

So in attention what we do? Now this is my encoder state right and this is the final hidden state of the encoder the first hidden state of the decoder this will have access to all the previous encoder states right and how do you do that? Here we introduce a concept of query and value. Each of this decoder hidden states will act as a query and each of the encoder hidden states will act as a value right. So the decoder hidden state will basically check which of the encoder hidden states will be relevant for that decoder state, right? Relevant means which encoder states will be helpful for the decoder state to generate the correct output. ok? how do you do that? So let us first look at the computation. So this is my query and this is value V.

So what I do, I'll first measure the similarity between query and value. And how do you measure this? We measure this using simple dot product. So this vector and this vector will be passed for dot product, right? And we get a scalar here. Scalar, this vector and this vector will pass through dot product, right and then similarly this is my query, this is v, So each of the decoder state will basically be used to do the dot product of every encoder state separately. So this is the second dot product, third dot product, fourth dot product.

What are these dot products indicating? They are indicating the similarity. Right, so these are called attention scores. All of these elements are scalars remember this. From attention score I pass them through shortmax. Shortmax will produce a distribution and this distribution is called attention distribution.

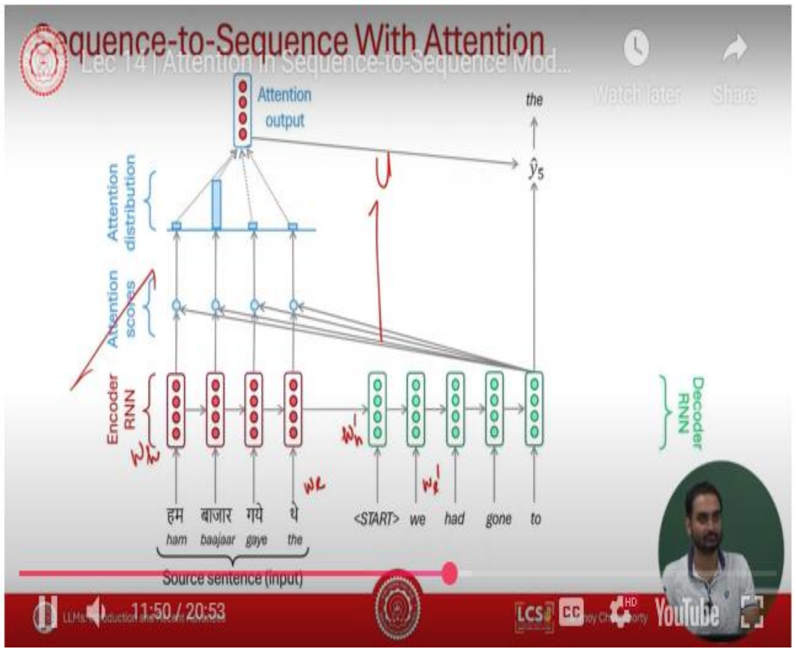
Let's say let's say this is p1, this is p2, this is p3 and this is p4. And let us say this is v1, v2, v3, v4, v1, v2, v3, v4 are vectors p1, p2 they are scalars and from there from this computation what I do? I will generate something called attention vector. Right what is the attention vector? Attention vector is the weighted sum of the encoder states and what are the weights, weights are the probabilities. So attention vector will be p1 and the corresponding vector v1 plus p2 v2 plus p3 v3 plus p4 v4 this times this plus this times this right and plus so on and so forth right. So p's are the weights.

So what is this vector? This vector is the weighted sum of the encoder state. And how does this weights are computed? The weights are computed based on the similarity between the decoder state and the encoder states right. Remember, these weights will be different when I move to the next decoder state, right. So I got this attention vector. I already have a hidden state here, and the attention vector here, which I compute So this is attention vector and this is the decoder vector decoder hidden state I just concatenate them.

So at this time, at time stamp 1, the hidden state is h1, right attention vector is let us say a1 I concatenate them right and then what I do? I pass it through a linear layer u followed by nonlinearity and it will generate a probability. So, instead of passing the hidden state directly to the linear layer, I will pass this concatenation of hidden state and attention vector right. So, similarly for the next decoder state, this is my q, and this is my attention distribution. Now look at this distribution of the previous distribution. They are different, right? Now if you look at the distribution carefully, to generate the first output token of the decoder, look at the probability.

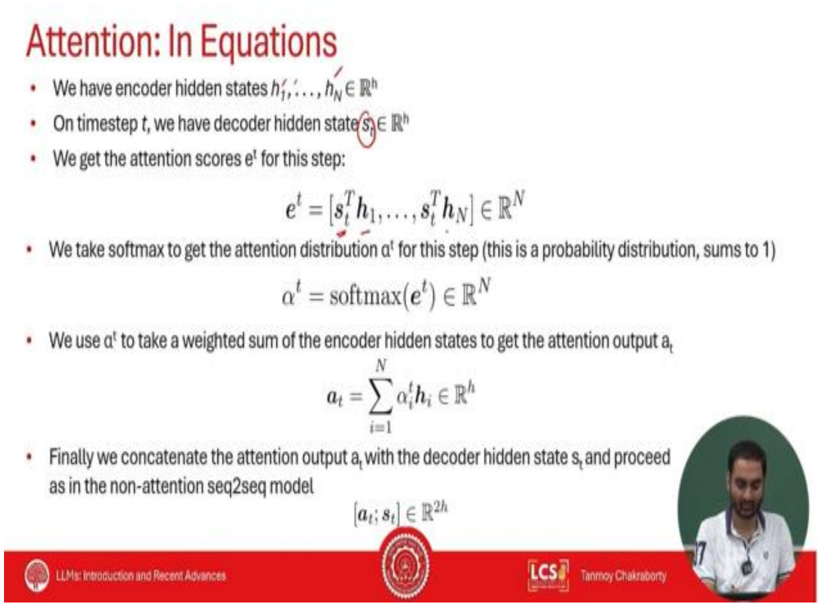
Which encoder was useful? Which encoder state was useful? The first one was useful. Therefore, the corresponding probability was high. To generate the second input had which one is useful the last one was useful, to generate the third one the third one was useful, to generate the fourth one the second one was useful right, and how do we get this usefulness from the probability dot product probability, this is clear, there is no bottleneck problem here. Why there is no bottleneck problem, now we have a shortcut from every decoder state to all the encoder states. Every decoder state can access all the encoder states.

Clear? So this is attention. Now tell me, due to this attention, due to incorporating the attention concept, did I incorporate any additional parameter? Answer me, quick. No. There is no parameter involved. What are the parameters here? The parameters are same. WH, WE, WH dash, WE dash and here U. That's all.



There is no U here. There is no U here. Attention score, attention distribution.

There is no U here. So this is vanilla attention. Okay. Of course there are modified versions of attention I will discuss but this is the vanilla attention.



Attention: In Equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution, sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

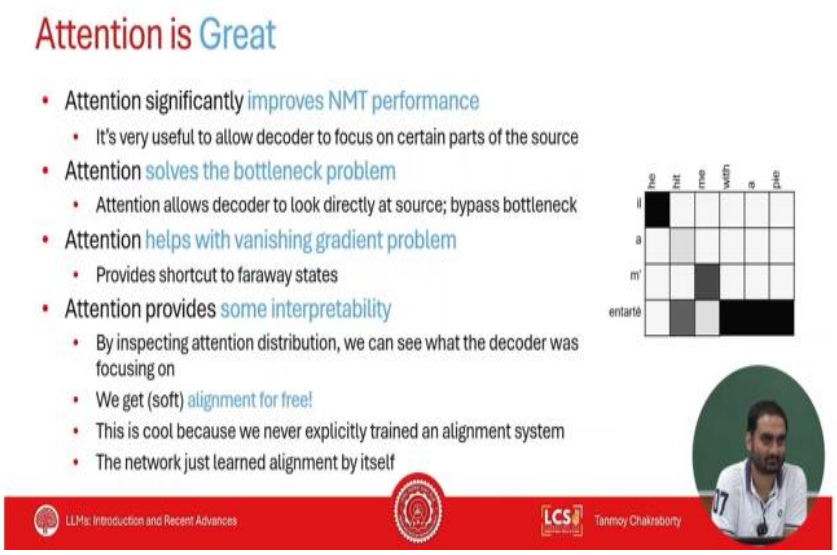
$$[a_t; s_t] \in \mathbb{R}^{2h}$$

LLMs: Introduction and Recent Advances   LCS2   Tanmoy Chakraborty

Okay, so let's look at the equation once again, the attention equation I have already mentioned. So h1 h2 h3, look at this equation right, h1 h2 h3 hn these are the hidden states of the encoder e right.

What is s t? s t is the hidden state of the decoder at the th time period. So, s t will basically access all the hidden states of the encoder so st h1 st dot product h1 right st dot product h2 and so on and so forth right. So we generate these numbers, these are dot products, these are attention scores, right. Let's say this is et attention scores will be passed to short max, so alpha t is the attention distribution which will act as weights right. So alpha t will then be act as a weight for each of these hidden states of the decoder.

So alpha i is i right t ranges from 1 to n. What is a t? a t is the attention vector right then a t and s t s t is the hidden state of the decoder this will be concatenated right and then this will be passed through the linear.

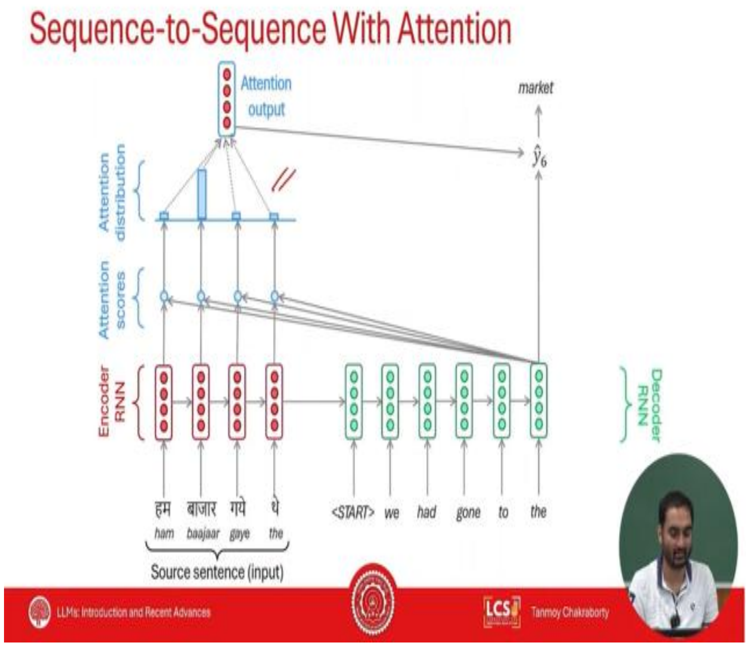


Attention is Great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

Attention is great why? Attention not only gives you a shortcut from decoder to encoder, it also gives you many additional information right. What are the information? It reduces the problem of bottleneck problems right i already mentioned okay bottleneck problem will not be there. It will also help address a vanishing gradient problem, why? Because now there is a shortcut right.

One gradient is moved through this, one gradient is moved through the shortcut, very similar to skip connection, right. The most important beauty of this attention is that it gives you interpretability.



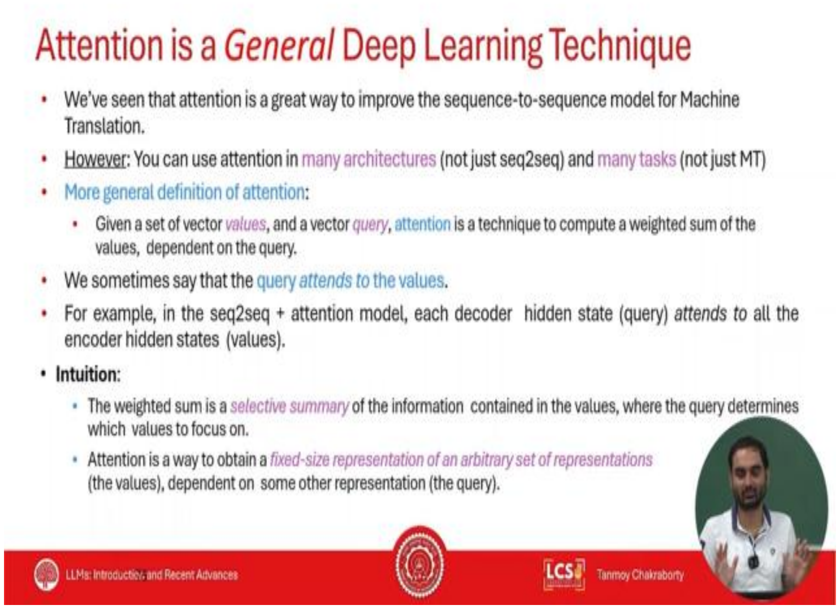
For example, here based on this distribution, you can say that to generate the last word of the decoder, second word was more responsible. You will not get this kind of information from a vanilla RNN. So if somebody tells you, can you explain that why you generated this, you can say that look at the probability.

So this is my attention map. So my french inputs and the english outputs and you see so this entry so each of these rows indicates okay attention distribution. If one il is dark it means the probability is high So, to generate the word he the word ill was responsible, to generate the word with a and pi the last word of the encoder was responsible therefore, start right. So, from the attention matrix you can identify the portions which are responsible. So, this will give you an alignment also. So, in a statistical machine translation model which I did not discuss.

There are three components. One is two components. One is the translation. So you basically, so I am going to the market, right? So it is not one-to-one translation. If you think about it carefully, this is not word by word translation. Sometimes we need to change things here and there.

So in statistical machine translation, there are two components. One is this one-to-one translation, and then you do some alignment. So alignment will rearrange the token, right? So for the alignment you need a separate model but here you do this alignment for free due to this attention. By the way, sometimes there are papers which claim that attentions are used for interpretability. There are papers which claim that attention cannot be used for interpretability.

This is debatable, but therefore I said it gives some interpretability.



Attention is a *General* Deep Learning Technique

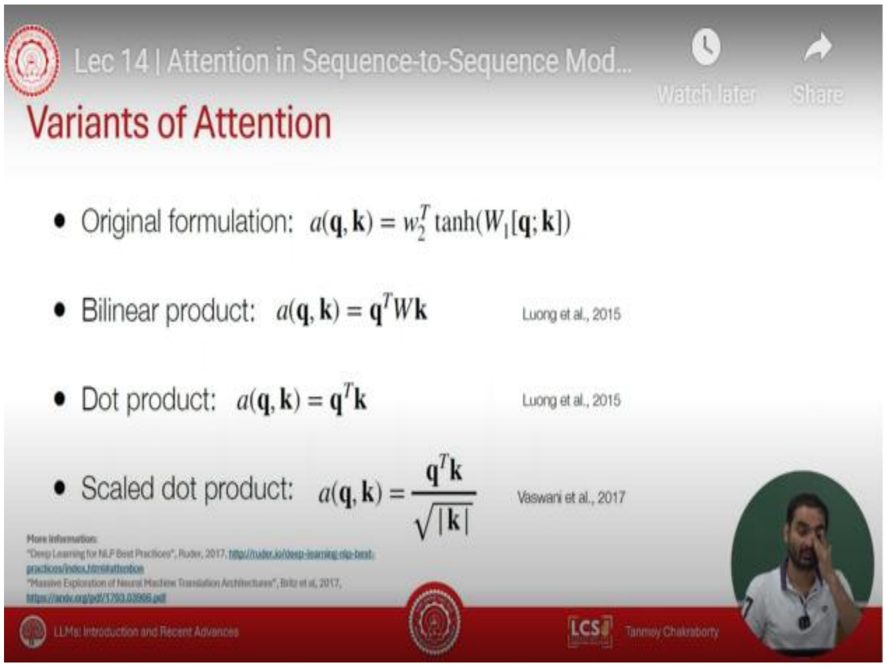
- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)
- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.
- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).
- **Intuition:**
  - The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
  - Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

So, I discussed attention with respect to sequence to sequence problem. But attention is not restricted to sequence to sequence problem. Attention can also be used in other models. You can use attention in normal RNN for example.

So in general you can think of this attention as a generic concept in deep learning where you have a query and you have some values and you want to measure the importance of query with respect to all the values, you can use attention. You can also think of an attention, the attention vector that you generated. Remember, attention vector is one vector that you generated, right? For every decoder, for every decoder industry, you generated one attention vector, right? So, attention vector can also be thought of as the summary of all the value vectors. Think about it. Attention vector can also be considered as a summary of all the value vectors.

In our case, the value vectors are encoded, encoded out hidden states. It's a summary, right? Irrespective of the size of the encoder layers, how many encoder layers? Let's say there's a sequence where there are 100 encoder layers there is another sequence where there are 10 encoder layers irrespective of the size of the encoder layers you can always generate one attention vector. So, it basically you can think of attention as a summary of the value vectors, you can think of attention as something which can squash the entire encoder states into a single vector. Why I said all these things? Because when you apply attention to some other problems, you need to understand all these factors.



Variants of Attention

- Original formulation: $a(\mathbf{q}, \mathbf{k}) = w_2^T \tanh(W_1[\mathbf{q}; \mathbf{k}])$
- Bilinear product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$ Luong et al., 2015
- Dot product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$ Luong et al., 2015
- Scaled dot product: $a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$ Vaswani et al., 2017

More information:
"Deep Learning for NLP Best Practices", Ruder, 2017. http://ruder.io/deep-learning-nlp-best-practices/index.html#attention
"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

These are some variations of attentions. The one that I discussed was a simple attention method. I basically discussed this one. This is called dot product attention. This is the one that I discussed. You can also think of some sort of parameter because we have a lot of data.

We will be very happy if we are able to add more parameters. You can also add a parameter between the query and the value, right. What you can do? You can pass the query to parameter matrix right you can also pass the value towards the same parameter matrix basically a bilinear product right and then it will generate some vector You can also think of doing some complex formula. For example, you concatenate the query and value, then pass it through a linear layer, some non-linearity, then pass it through another linear layer.

That will give you a vector. The transformer paper, they use something called scaled attention, scaled dot product. where we take the dot product and this dot product is normalized by the size of the vector. What is mod k? Mod k is the size of the value and you take the root of this. So, this is scaled by the size of the vector.

There is no intuition by the way. I mean these are different hacks, engineering hacks. Okay, so this is all about attention. We stop here and in the next class, we will discuss self-attention and then the transformer model. Okay, thank you.