Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

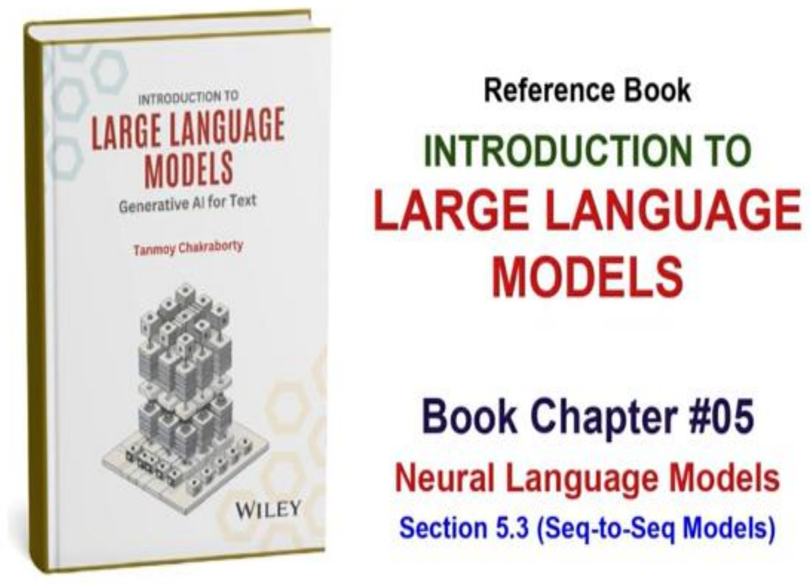
Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

Lecture 12

Sequence-to-Sequence Models

Hello everyone, in today's class we will discuss sequence to sequence modeling. In the last class we discussed RNN methods, advanced RNN methods, LSTM, GRU and so on and so forth. We have seen how gated, gating mechanism can address the vanishing gradient problem. We will see how RNN can be used in the other types of problems like sequence to sequence labeling problems.



Reference Book

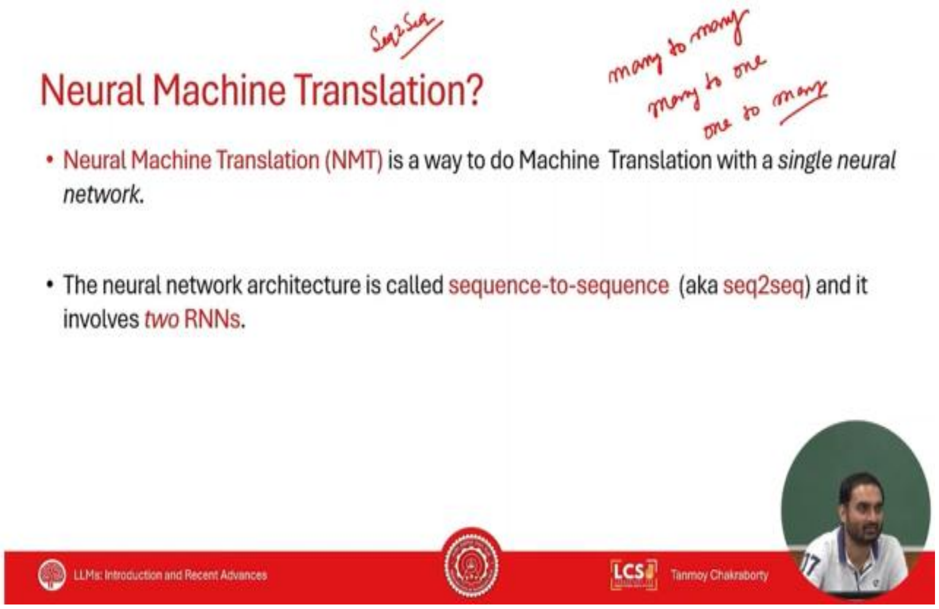
INTRODUCTION TO

LARGE LANGUAGE MODELS

Book Chapter #05

Neural Language Models

Section 5.3 (Seq-to-Seq Models)



Neural Machine Translation?

Seq2Seq

many to many
many to one
one to many

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network.*
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

Let us get started. So sequence to sequence models, we are discussing those things which are proposed in 2015, 2016.

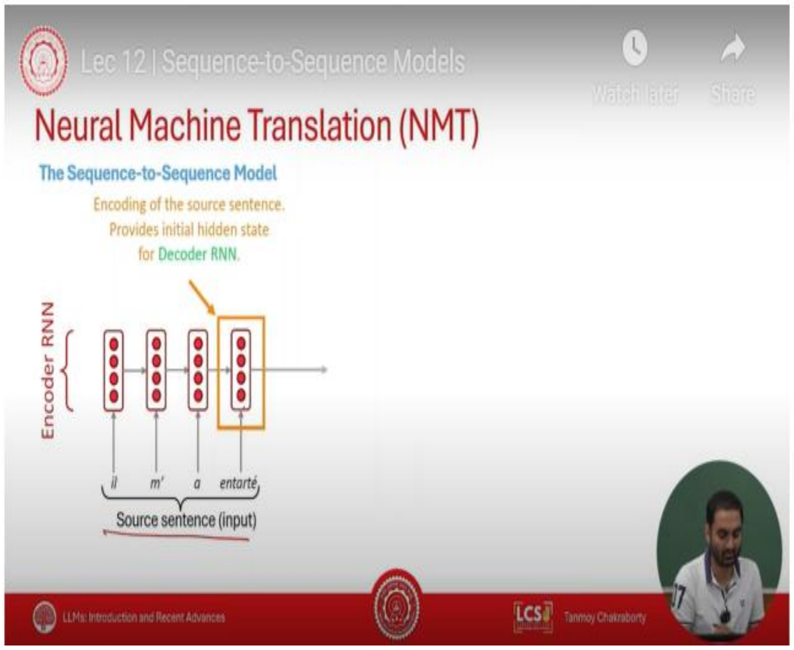
So in sequence to sequence model, what is a sequence to sequence problem? A sequence to sequence problem is you are given a sequence and the output would also be a sequence. Right let's say machine translation you are given a hindi sentence and the output would be english sentence. Let's say summarization you are given a longer document and your task could be to basically generate a shorter document right. These are all sequence-to-sequence problems, right.

It can be many-to-many, it can be many-to-one, it can be one-to-many and so on and so forth. Many-to-many sequence to sequence model, sequence to sequence the term can also be abbreviated as sec to sec. So, many to many sec to sec problems include summarization, dialogue. Many to one, what do you think? Many to one sec to sec problem can be classification. No.

So it's not a sequence to sequence as such. Because ultimately you are generating a label. But you can think of it as sequence to sequence. It also depends on the way you map it. What is one to many? Example of one-to-many? No.

Speech is a sequence to sequence. Speech generation. Music generation. What's the input? Input will be a prompt right or so that is many to many. What about image captioning? Your input is an image and output is a text, a sequence.

Of course, you can say that I can basically flatten the image matrix or whatever that will be a many to many, but high level this is one to many.

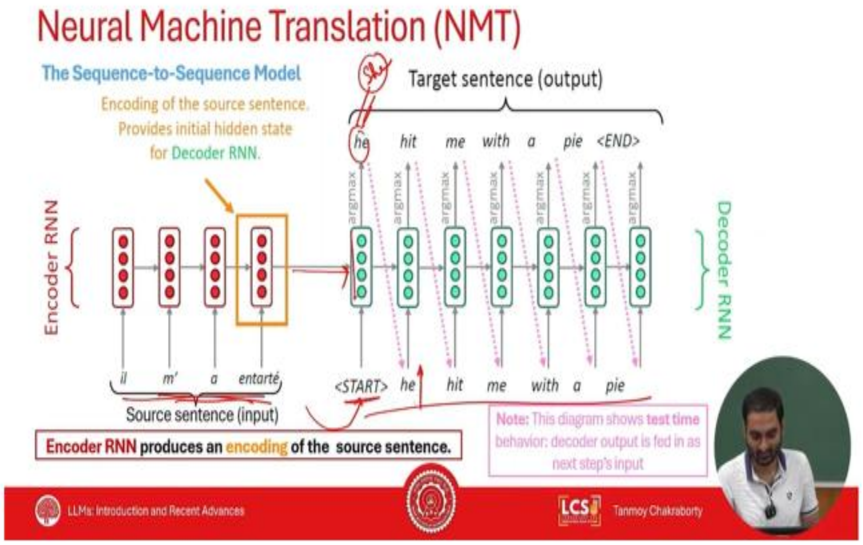


Lec 12 | Sequence-to-Sequence Models

Neural Machine Translation (NMT)

The Sequence-to-Sequence Model

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.



Neural Machine Translation (NMT)

The Sequence-to-Sequence Model

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.

Encoder RNN produces an encoding of the source sentence.

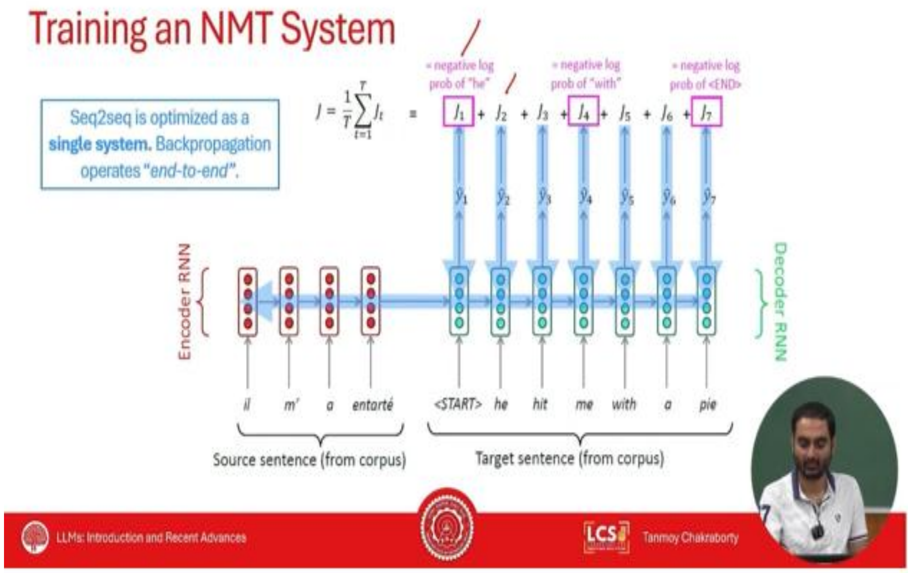
Note: This diagram shows test time behavior: decoder output is fed in as next step's input

So, let us look at this machine translation problem for the time being. In a sequence to sequence model what we do? Since there are two sequences involved input and output, we will use two RNNs. Right one RNN will process the input another RNN will process the output okay Let's say this is the french sentence, I'm not able to pronounce it at all. I don't know french.

Let's say this is the French sentence and you use a simple RNN right to encode the the sequence and let's say this is the last hidden state right and this is your output. The translated version of this French sentence is "He hit me with a pie," okay. So the last state of the previous RNN will be fed into the first state. The output of the RNN and the remaining process will follow. Now, if you look at this thing very carefully, Where is the loss we are computing? So we are computing the loss at the hidden state output of the RNN.

This is the output RNN, right? Now, look at the time stamp. So here, let's say when you allow, this is the input, right? Ideally, you know the first input of the first hidden state will be a start symbol, right? And it should be able to generate the next word, which is 'he.' Right. Now, you pass this 'he' to the next hidden state; it will generate something. Yesterday, I mentioned something called teacher forcing, right? What is the teacher enforcing? In teacher forcing, instead of feeding 'he', let us say here that the model generates 'she'.

So, ideally I should feed 'she' and then I will see what the next What is going to be right? This is a normal process, but in teacher forcing, I will feed the actual input correctly. For example, even if it generates she, I will feed him in the next state, right? Which is kind of cheating, but this is it, I mean. It turned out to be much more effective in at least training the module. During inference, you can't do this, but during training, you can do well. So at this stage, you will have some error and so on and so forth, right?



Training an NMT System

Seq2seq is optimized as a **single system.** Backpropagation operates *"end-to-end"*.

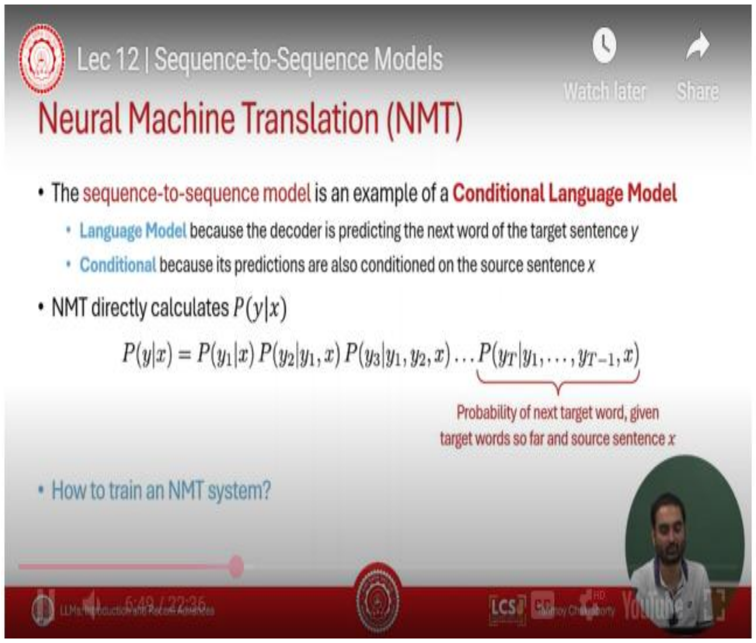
$$J = \frac{1}{T}\sum_{t=1}^{T} J_t = \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$$

= 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

Look at this. So here you have an error: j1, j2, j3, j4, j5, dot, dot, dot, j7, right? And the total error is basically the sum of this. This is the total error, and you normalize it by the number of output tokens, t. And then when you backpropagate, when you do backpropagation right. so this loss will affect all the hidden states.

This is end-to-end training. This is called end-to-end training, okay. So this error will be back propagated and all the hidden states in the decoder as well as encoder will be updated. So the input RNN is called the encoder, and the output RNN is called the decoder. This is end-to-end training.

Clear? Okay. And how do we do this backpropagation? We will use BPTT, backpropagation through time. Okay.



Lec 12 | Sequence-to-Sequence Models

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - Language Model because the decoder is predicting the next word of the target sentence y
  - Conditional because its predictions are also conditioned on the source sentence x
- NMT directly calculates $P(y|x)$

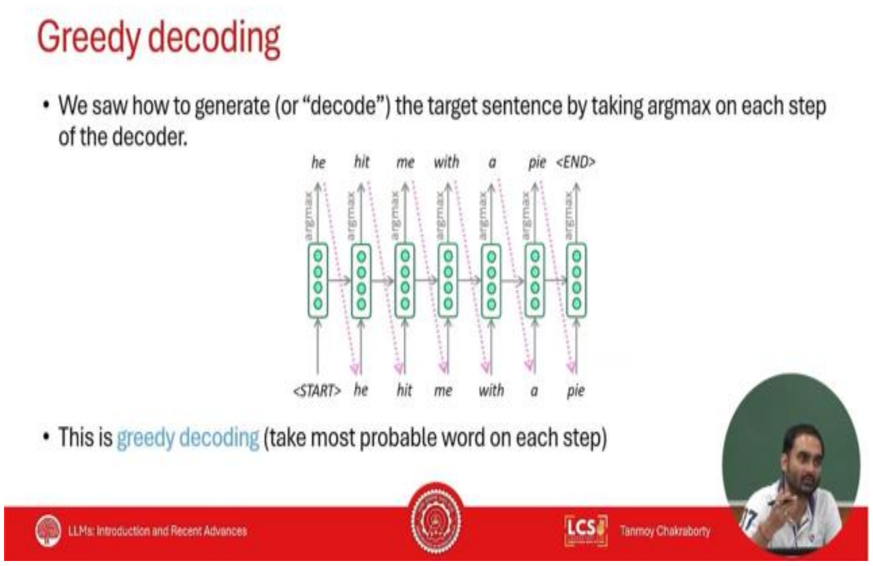
$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots\underbrace{P(y_T|y_1,\ldots,y_{T-1},x)}_{\text{Probability of next target word, given target words so far and source sentence } x}$$

- How to train an NMT system?

Now, if you think about this sequence-to-sequence problem carefully, This is essentially a conditional language model. Why is this a conditional language model? What is it conditioned on? It is conditioned on the correct input. So, let us say x is the input here.

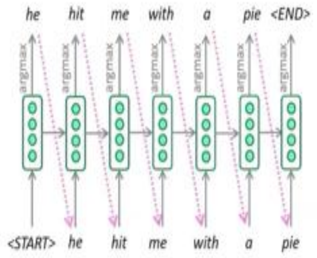
Input is, in the case of machine translation, the French sentence. And the output is the English sentence, right? So, you are basically computing this probability. So this is now, if you think of y as y1, y2, y3, dot, dot, dot. So the first probability that you are computing at the first hidden state is correct, right? What is the input there? The input is the final hidden state of the encoder, isn't it? So that is X. But you have already processed the entire input sentence, right? So, the probability of y1 given x, the next one, this one, What is this one? This is the probability of y2 given x and y1, right? And so on and so forth.

The last word will be the probability of yt. Given y1, y2, y3, ... , yt minus 1 and x right. So condition on the generation the tokens that you have generated so far And the input token is okay.



Greedy decoding

- We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder.



- This is greedy decoding (take most probable word on each step)

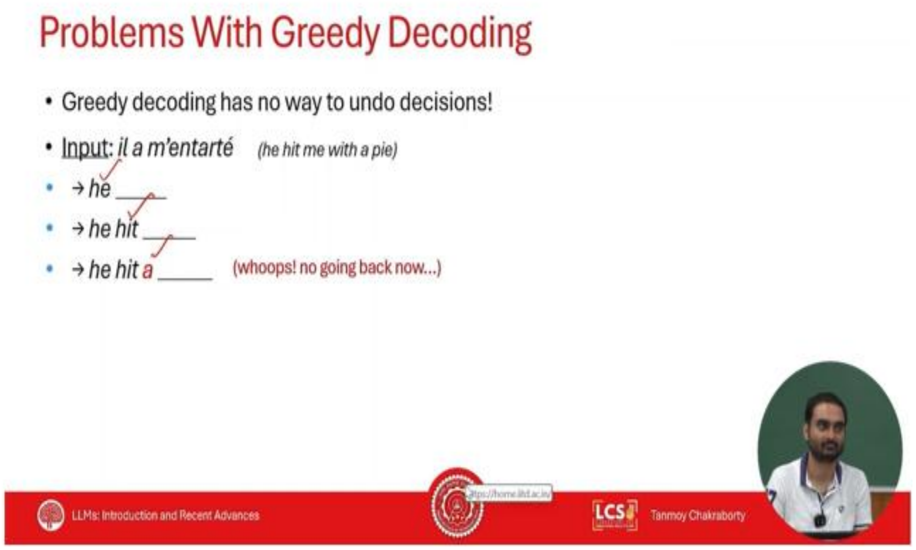
So, now when we decode this, how do we decode this? During the inference stage, can somebody tell me, let's say the model has been trained, okay? How is the model trained? I think it is clear, isn't it? How is the model trained? We will do teacher forcing, right? And we will compute the loss at every time stamp, right? And then we backpropagate the entire loss end to end, okay? And you do it for all the machine translation pair. Now, during the inference stage, what do you do? You are given a, now since there are two in a variance encoder and decoder, How many weight matrices are there? So in the encoder, how many weights are there? There is this WH, right? There is this WE, and there is this U. So U is not present in the encoder. Are you there or not? What is U? The U matrix was the linear projection matrix, right? That is not in the encoder because, at the encoder stage, You are not printing anything.

So you need a wh and a we. What are we? we are the weight matrix corresponding to the embeddings. And at the decoder stage, what are the matrices? We and these matrices are trained. or learned during the training period. Remember, the embeddings that you are using here are different. Because the input is a French sentence, the output is an English sentence.

The embeddings are different, right? Let's see if it is a summarization problem. Then the embeddings will be the same because the languages are the same, okay? So now, during the inference time, what will we do? We are given the French sentence; we pass it to the encoder, right? We will use these matrices; using these matrices, we will get the. final hidden state of the encoder right, that hidden state will be given to the decoder and in the decoder I already know We dash right. So, based on that, I will generate this hidden state, the first hidden state. Of the decoder right, and then I pass it through this linear layer u called by nonlinearity. It will project it to a one-million-dimensional vector. One million is the size of the vocabulary, right? Now it will project it to a one billion-dimensional probability vector. Then you sample one token from the probability distribution, and the hope is that. The token that has the maximum probability will be sampled. So let's say you sampled he okay and then you feed here It will generate a hit if you feed.

So when I say I feed him to the next layer, what does it mean? It means I feed the embedding of him to wh. This wh-dash is the same for all the tokens that. you are generating in the decoder. In a greedy decoding strategy, instead of sampling the token based on probability, I will choose the token that has the maximum probability.

I will not do any sampling. This is greedy decoding.



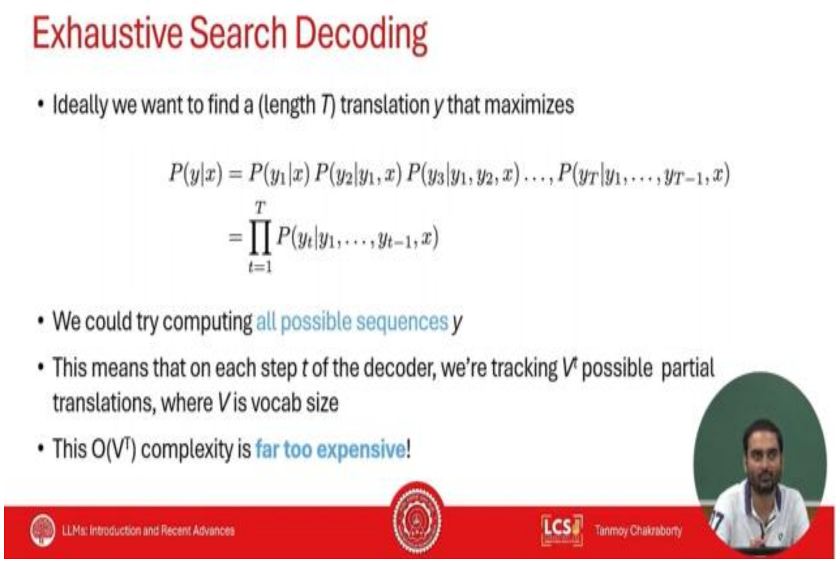
Problems With Greedy Decoding

- Greedy decoding has no way to undo decisions!
- Input: *il a m'entarté* *(he hit me with a pie)*
- → *he* ____
- → *he hit* ____
- → *he hit a* ____ (whoops! no going back now...)

So, what is the problem with greedy decoding? The problem is that, let's say you made a mistake at any point, You won't be able to recover it. Let's say you want to generate "I hit." So you wanted to generate, "He hit me with a pie.

" So you generated him. And then you generated a hit. And suddenly you generated a. But you should have generated me. Isn't it? You won't be able to backtrack.

Because it has already been generated. The A will be fed to the next state. And in the next state, something else will be generated. This is the problem with greedy decoding.



Exhaustive Search Decoding

- Ideally we want to find a (length *T*) translation *y* that maximizes

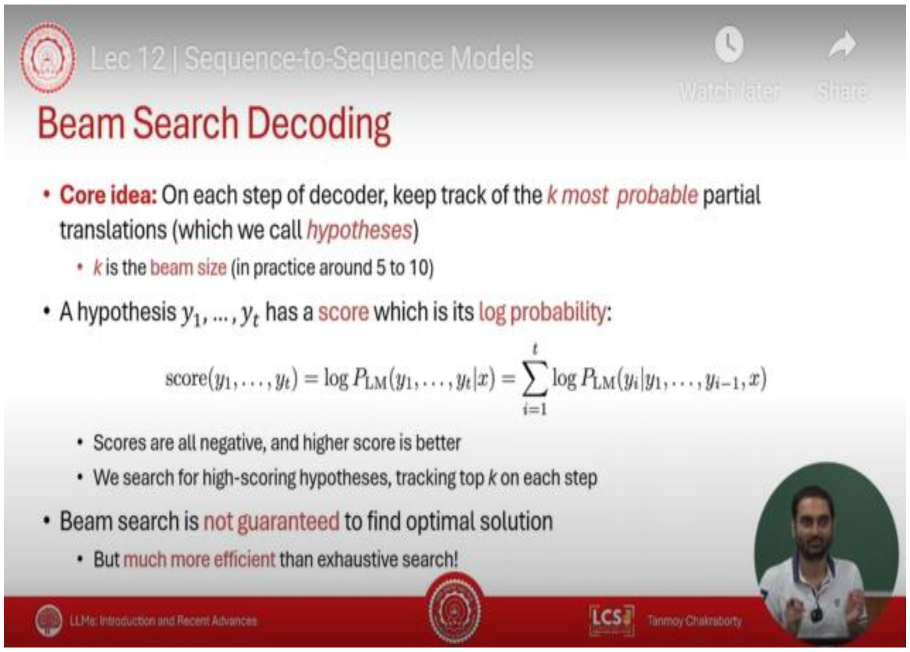
$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

- We could try computing all possible sequences *y*
- This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
- This $O(V^T)$ complexity is far too expensive!

Now tell me, what do you think? How do you address this problem? Is it possible to conduct some sort of exhaustive search to address this problem? So instead of, now think about it, instead of generating one token at a time, If I could have generated V number of tokens, what is V? V is 1 million.

I generated all V number of tokens. Now, again for each of these V tokens that I have generated, I pass it to the next layer. It will generate V tokens again, and so on and so forth. V tokens, and so on and so forth. Is it possible to do that? It is not possible to do that. Let's say, if it is possible to do, what would the complexity be? There are T number of timestamps in the decoding stage.

Let's say V number of possible tokens that you can generate. How many possible branches do you need to keep track of? V to the power T, this is extremely exhaustive. Okay, so how do we solve this problem? Let's not make it that exhaustive. Let's squeeze it as much as we can and basically squeeze the entire thing. We will use something called beam search, okay? In beam search, what do we do?



Lec 12 | Sequence-to-Sequence Models

Beam Search Decoding

- **Core idea:** On each step of decoder, keep track of the *k most probable* partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)
- A hypothesis $y_1, \dots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step
- Beam search is not guaranteed to find optimal solution
  - But much more efficient than exhaustive search!

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty

The idea of beam search is the same as that of exhaustive search.

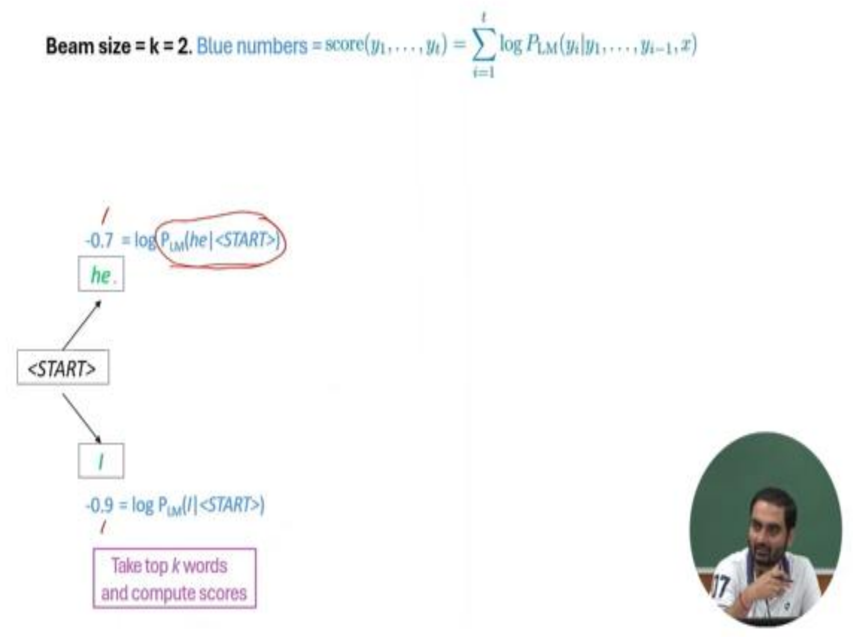
The only difference is that instead of keeping track of V possible options, I will only keep track of K possible options. Where K is the size of the beam. Right, it means that instead of generating only one token as we did for greedy decoding, For v tokens, as we did for exhaustive search, I will generate k tokens. k can be 5 or 10, whatever you set the value of k to, okay. So at every stage here, let's say I generate k tokens, Now, from each of these k tokens, I further generate k tokens and so on and so forth. So the complexity is still significantly high. Complexity is still k to the power of t, isn't it? But k is much, much smaller than v, right? And you can decide the value of k, okay? So now what we do is think about it. You generate a token here at time t. You generate a key number of tokens.

Token 1, token 2, key number of tokens here. And from each of these tokens, you further generate a key number of tokens. and so on and so forth. And then you stop at a certain point. When do we stop? When do we stop the decoding phase? When do we stop? I will give a French sentence when this decoding stage stops.

When it generates a token called EOD. There are two tokens: special tokens start and end of sentence. or whatever end of decoding eod right, full stop basically, right. So this is one way of basically stopping the decoding phase; there are other ways, too. So, each of these branches, right, So let's say this branch will lead to the end of the token.

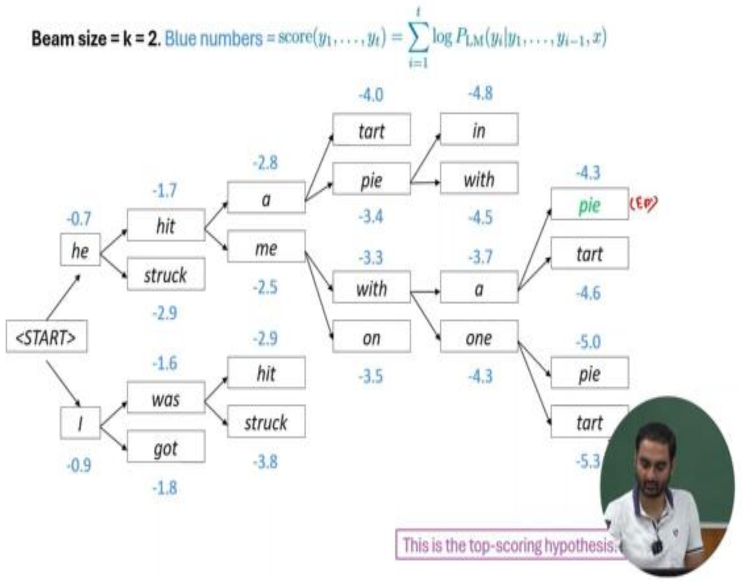
This branch will also lead to end of token. This branch will also lead to end of token. So there are multiple branches, aren't there? Each of these branches is called a hypothesis. Why the hypothesis? Because we do not know which of these hypotheses is correct. Which of these branches is correct? Therefore, this is called a hypothesis, right? So now which hypothesis will I choose at the end? I will choose a hypothesis whose score is higher. And how do we measure this score? This code is essentially the log probability of the sequence.

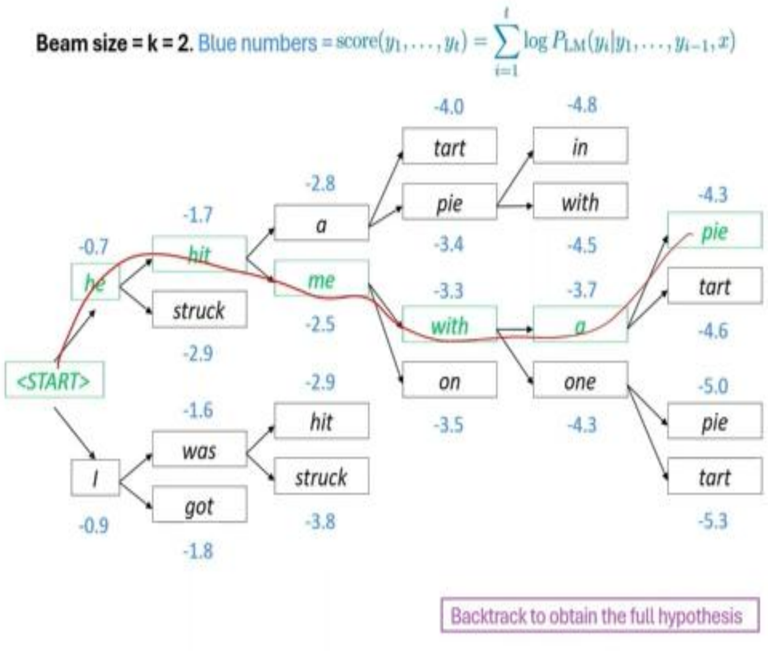
That we have generated so far is clear. So, two things I mentioned: one is the value of k, right? We decide on a k, and then every hypothesis will be assigned a score.



Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

-0.7 = log P_LM(he|<START>)
he
<START>
I
-0.9 = log P_LM(I|<START>)
Take top k words and compute scores


Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$


Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

Okay, now let's look at an example. This is an example. So a beam starts off size two; I start with this special token 'start'.

This is the decoding stage inference time. not the training time remember this. Let's say the value of k is two, so I generate 'i'. I have generated two tokens: 'i' and 'he'. Right, and what is this probability, this log probability? How do we obtain this probability? We will get the probability from the distribution, won't we? Because we have this linear layer and then softmax, right? We will get the probability from the distribution, and we will take the log probability. Log probabilities basically, we sum all the log probabilities, which is the same as multiplying all the probabilities.

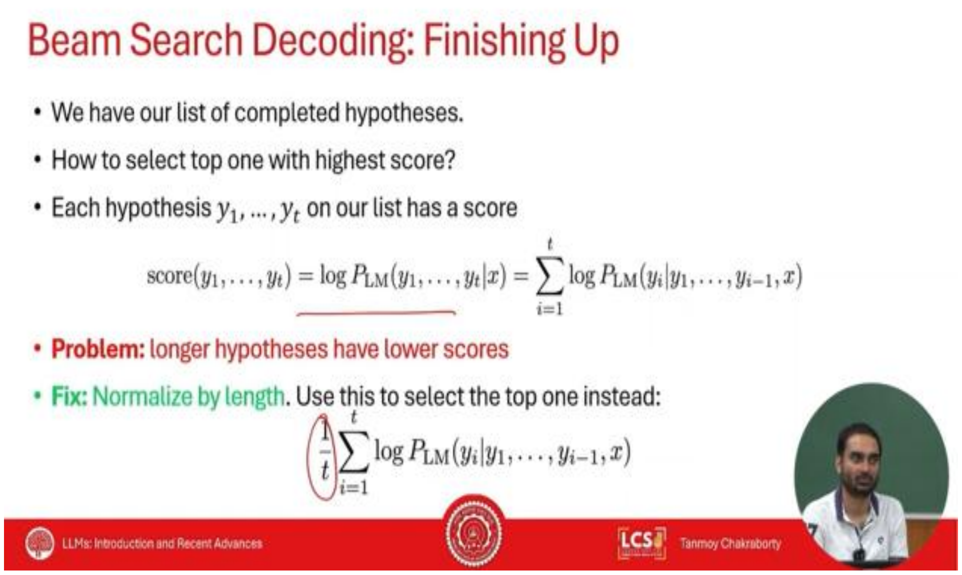
Now, since this is probability, it will range between 0 and 1. The log of this will be negative right. So all these negative numbers will be added, and I will choose that hypothesis. whose value is maximum, which is less negative.

Here, the score up to this point is 0.7 minus 0.7 minus 0.9. Which one will I choose? Which one will I choose? I don't need to choose because the value of k is already 2. Right, let's explore this further. Now from 'a', from 'he', the next two tokens can be hit and started right. From 'i', the next tokens can be "was" and "got." So, till this part, what is the probability t? So at this stage, the probability is the sum of this and the previous one.

So, the total probability is -1.7. Here, the total probability is -2.9, -1.6, -1.8. So, out of this, how many will choose? We chose only two. So, which two will you choose? We will choose the highest one, right up to the maximum one. So, one is minus 1.7; the other is minus 1.6. Right, and then we explore further. It will generate further; from it, I will generate A and me. From that, I will generate hit, struck, and so on and so forth. Here again from this 4, I will decide on 2 right.

Now look here. This branch is gone wrong. Now I am only exploring the top branch. From each of these again, I will generate two rights, and so on and so forth. Again, these two will be selected; from these two, I will generate two from each. these two will be selected and this right, Let us assume that hereafter the EOD is generated. Okay, so among these, which one is the maximum? The maximum one is this one minus 4.3. Then I will backtrack, and this is the backtracked right. So I will generate this. The problem here is that longer hypotheses will be penalized. Shorter hypotheses will be given more advantage; why? If it's a longer hypothesis, then you keep adding multiple minuses. Or basically, you are multiplying so many probabilities, which will lead to.

You know the lower probability value. Let's say you have one hypothesis where there are only two tokens versus You have a hypothesis in which there are 10 tokens. So, which one will be preferred? Let's say both these hypotheses successfully led to an end of the token, right? Which one will be preferred? The shorter one will be preferred because of this right.



Beam Search Decoding: Finishing Up

- We have our list of completed hypotheses.
- How to select top one with highest score?
- Each hypothesis $y_1, \dots, y_t$ on our list has a score

$$\text{score}(y_1,\dots,y_t) = \underline{\log P_{\text{LM}}(y_1,\dots,y_t|x)} = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\dots,y_{i-1},x)$$

- **Problem:** longer hypotheses have lower scores
- **Fix:** Normalize by length. Use this to select the top one instead:

$$\frac{1}{t}\sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\dots,y_{i-1},x)$$

So what I do here is normalize the score. By the number of tokens it has generated.

So if a longer token is generated, I will normalize it. By the number of tokens, a shorter token again has a number of tokens. that will nullify this one. Thank you.