Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

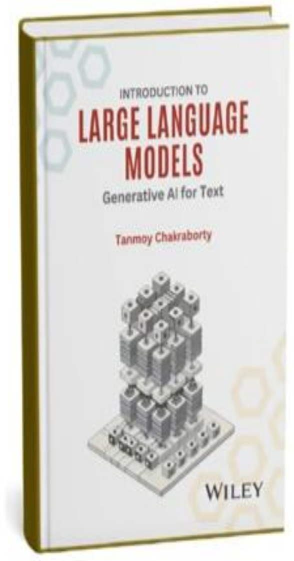
Lecture 11

Neural Language Models: LSTM & GRU

In today's lecture, we will continue our discussion on neural language model. So, in the last class, we started discussing vanilla RNN, we discussed the architecture of RNN and then the training of RNN using backpropagation through time, PPTT. And we have also seen that due to the recurrent multiplications of gradients, what happens is that if the size of the gradient is small, it would lead to vanishing gradient problem. If the size is large, it would lead to excluding gradient problem. So, although theoretically, RNN can capture long context but in practice, it starts forgetting those context which are far from the current context, current word. So you know exploding gradient can be addressed using methods like gradient clipping right whereas vanishing gradient can be addressed in various ways.

So in today's class we will discuss different gating mechanisms, we will discuss LSTM and GRU right. But in the next lecture maybe we will discuss more advanced topics like attention right. So attention LSTM, GRU these are different ways, but apart from these techniques there are also other ways for example, you can add skip connection, residual connection etcetera. We will discuss all these concepts in today's lecture.

Let us get started.



Reference Book

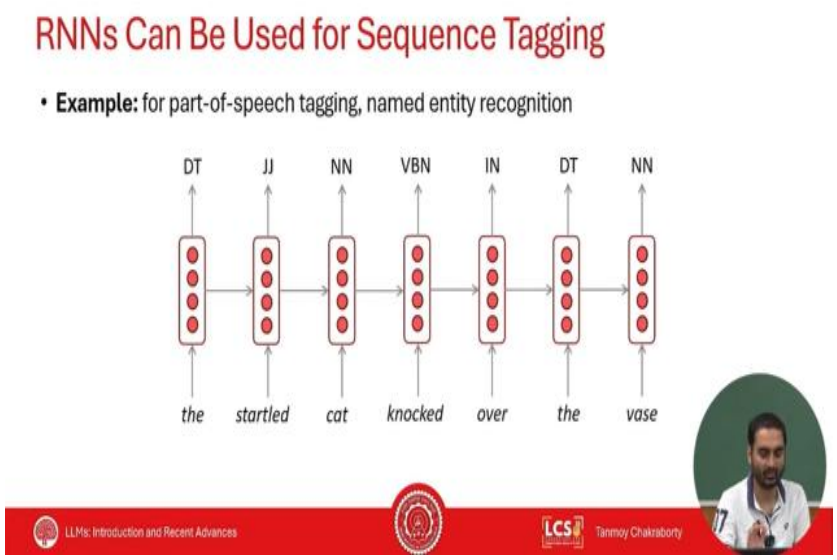
INTRODUCTION TO
LARGE LANGUAGE MODELS

Book Chapter #05

Neural Language Models

Section 5.2.4 (RNN Variants)

Uses of RNNs



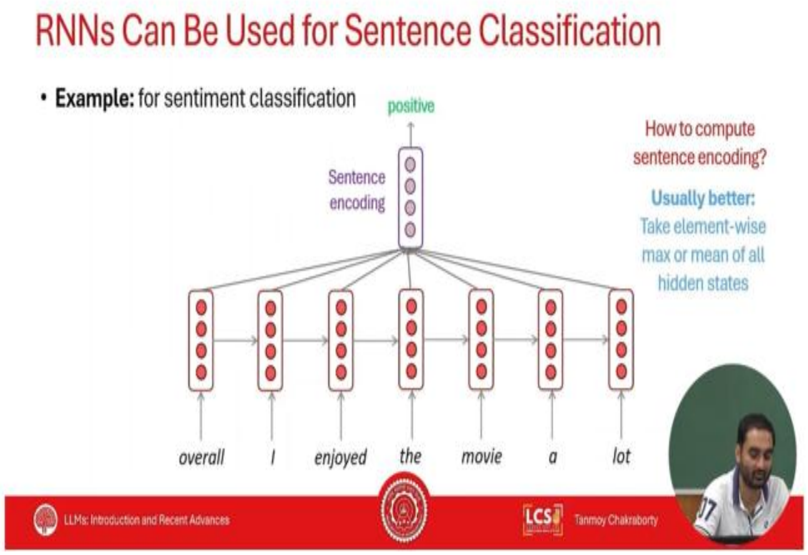
RNNs Can Be Used for Sequence Tagging

- **Example:** for part-of-speech tagging, named entity recognition

So, we will first look at some of the usage of RNNs in NLP in language technology. Since RNN is a sequence labeling method, given any sequence it should be able to solve downstream tasks. Let us say the downstream task is a task like partial speech tagging, POS tagging. So, you are given this sequence. So, this is the sequence. and you are asked to essentially determine the POS tag for every token in the sequence, right. So, what you do? You run RNN, vanilla RNN, right. So, let us say there are 46 tags, 46 POS tags, right. So, from every hidden state, you pass it through a linear layer U, last day we discussed, followed by a non-linearity, short max, etc. and short max will essentially map this to a 46 dimensional vector. and you choose the one which has the highest probability and so on and so forth. We choose the one which has the highest probability, not we choose the one, we sample it from the distribution. Remember this, because this is not a generation task, this is a classification task. And you do this for all the words.

Now, here you can ask that in order to predict, let us say, the POS tag for the word cat, at this hidden state, I have access to only the previous hidden states, but sometimes when we do POS tagging or let us say named entity recognition, we may also need to access to the future hidden states. and remember the entire sequence is given, it is not a generation task. The entire sequence is given to you and the task is to predict the POS tag for every word, right. So why only to consider the previous hidden states, we can also consider the future hidden states, right. So we will discuss in today's class that here we will use bi-directional RNN, there is one RNN from left to right, there is another RNN from right to left.

So at every time stamp you will have two hidden states, one from left to right and other from right to left. You have two hidden states and then you can either concatenate or you do some operations and basically create a single hidden state and from the single hidden state then you pass it through linear layer and non-linearity. This is called bi-directional RNN.



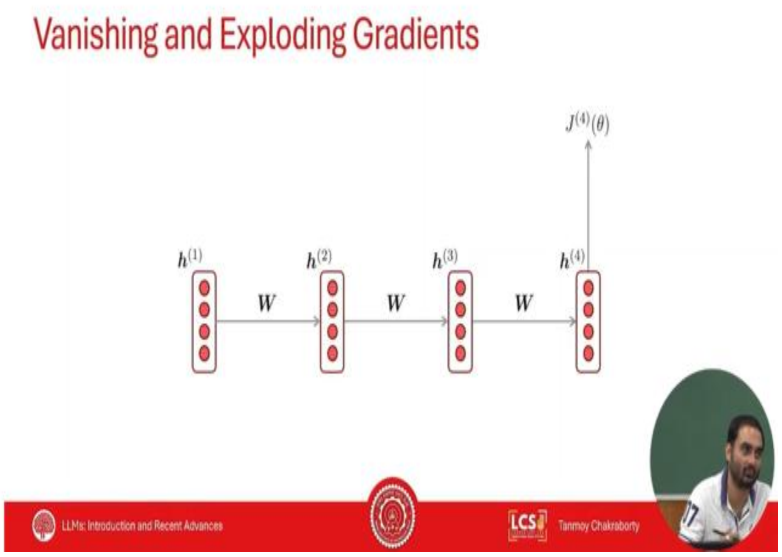
RNNs Can Be Used for Sentence Classification

- **Example:** for sentiment classification

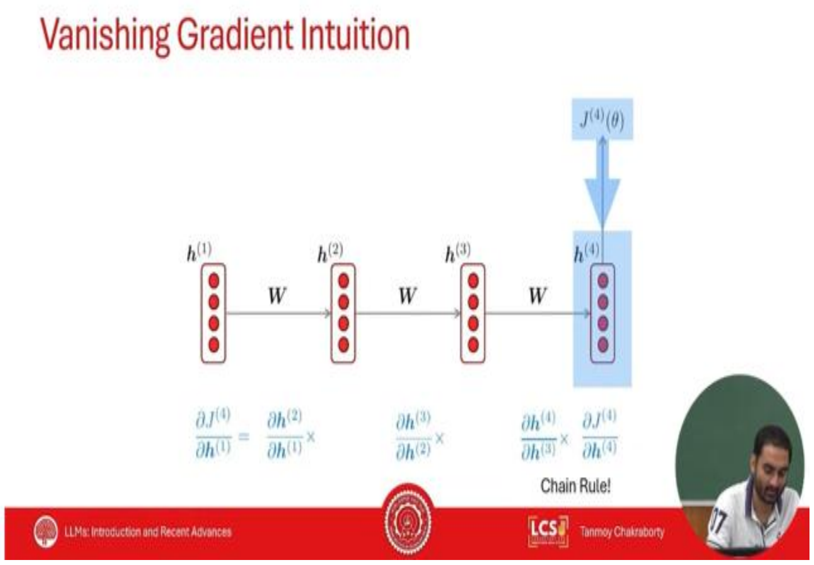
positive
Sentence encoding
How to compute sentence encoding?
Usually better:
Take element-wise max or mean of all hidden states
overall
I
enjoyed
the
movie
a
lot
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

You can use RNN for simple sentence classification task for example, sentiment analysis for example or let us say spam email classification. So what you do here, again there are different ways to do it.

One way would be you look at this sentence, there are hidden states here, one RNN, right? Then you basically merge all the hidden states together some ways and you create a single vector. This can be possible. Another possibility is that you look at, you take only the last hidden state because the assumption is that the last hidden state has information to all the previous hidden states. So you take the last hidden state and you pass it through linearly and non-linearity and so on so forth right.



Vanishing and Exploding Gradients



Vanishing Gradient Intuition

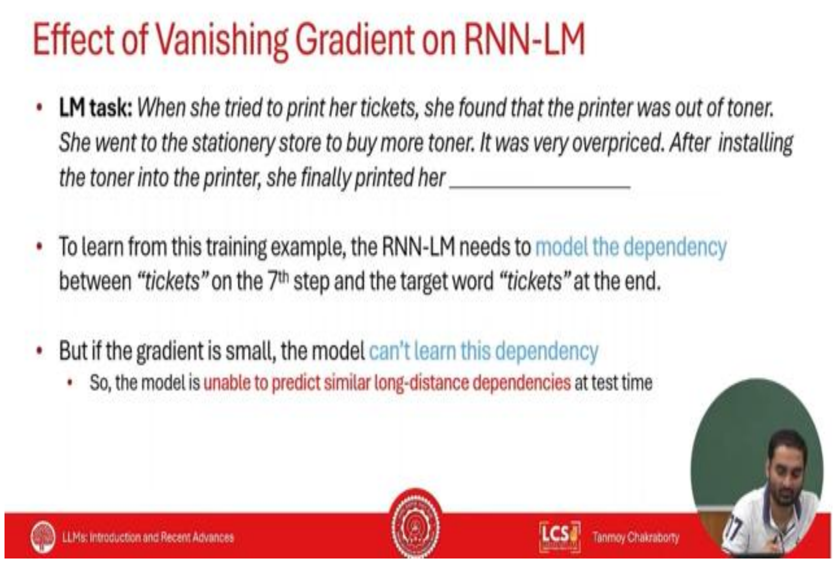
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

Chain Rule!

So, the problems with RNN I mentioned last day vanishing gradient problem, excluding gradient problem and it happens due to this recurrent computation of gradients right.

Let us say this is the last loss of the final loss which is J theta right at the fourth step right. I discussed this thing in the last class ok. If you want to take the gradient of this with respect to H1, you will have to take the gradient of this H4 with respect to H3, H3 with respect to H2, H2 with respect to H1 and so on and so forth. Right and if so these gradients right these are essentially matrix because you are taking the derivative of a vector with respect to another vector ok. Now if the size of this matrix is small right then it will lead to gradient vanishing gradient problem with size is large it will lead to exploding gradient problem.

Okay so and how to address this problem we discussed last day.

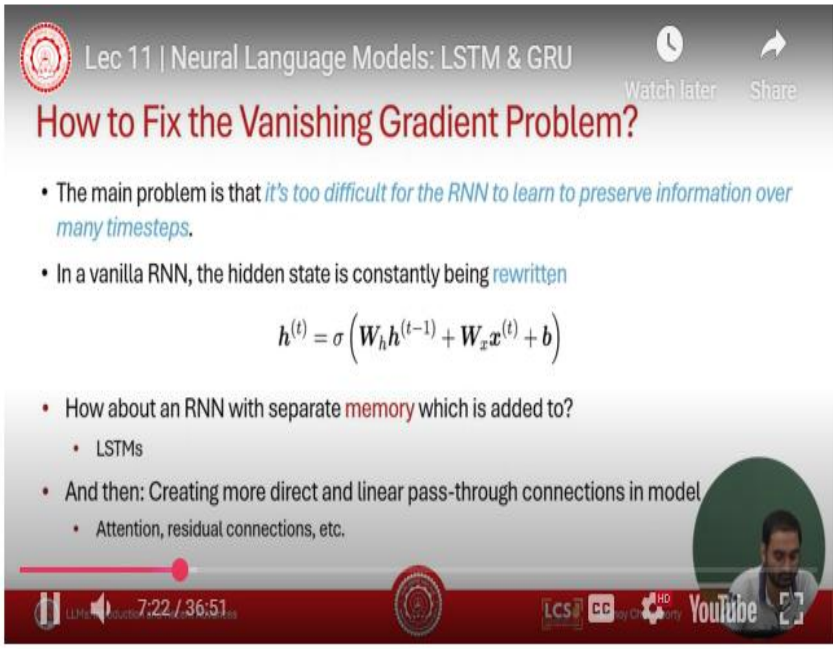


Effect of Vanishing Gradient on RNN-LM

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her ________________*
- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.
- But if the gradient is small, the model can't learn this dependency
  - So, the model is unable to predict similar long-distance dependencies at test time

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

Now how to address this we will discuss today but let us look at the problem here why this is a problem right. Let us look at this sentence right. So when she tried to print her tickets she found that the printer was out of toner. She went to the stationery store to buy more toner it was very overpriced after installing the toner into the printer she finally printed her dash right This will be tickets Now look at the distance between this and this, tickets, what tickets in that this is I think at the seventh position of the sentence and this is very very far so vanishing gradient Due to vanishing gradient problem RNN would not be able to predict the next word's tickets.


Lec 11 | Neural Language Models: LSTM & GRU

How to Fix the Vanishing Gradient Problem?

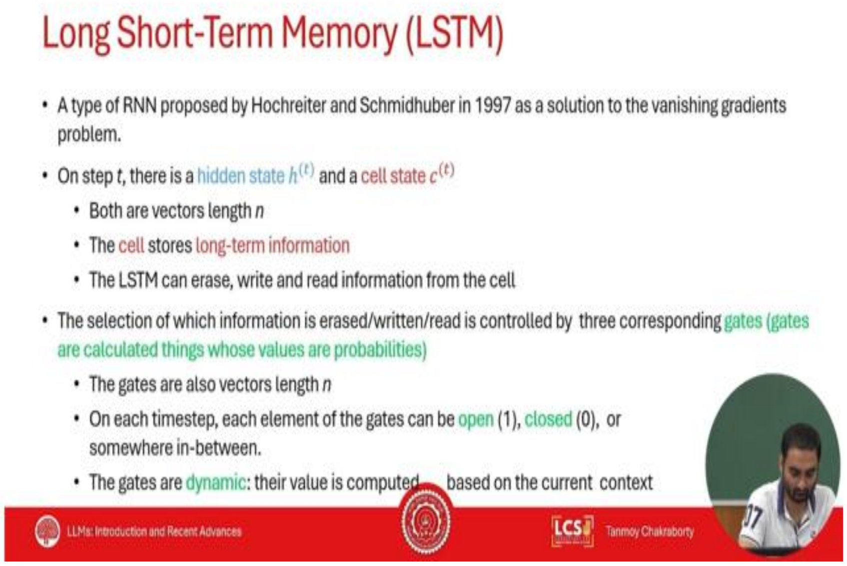
- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.
- In a vanilla RNN, the hidden state is constantly being rewritten

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$$

- How about an RNN with separate memory which is added to?
  - LSTMs
- And then: Creating more direct and linear pass-through connections in model
  - Attention, residual connections, etc.

So what are the ways we will fix it using a gating mechanism. We will see that in a gating mechanism we will explicitly use another state called the memory state. So along with the hidden state we also have a memory state called the cell state, we will discuss. So using LSTM, GRE we will try to address the problem of vanishing gradient but there are other mechanisms also like attention, residual connection we will discuss later.

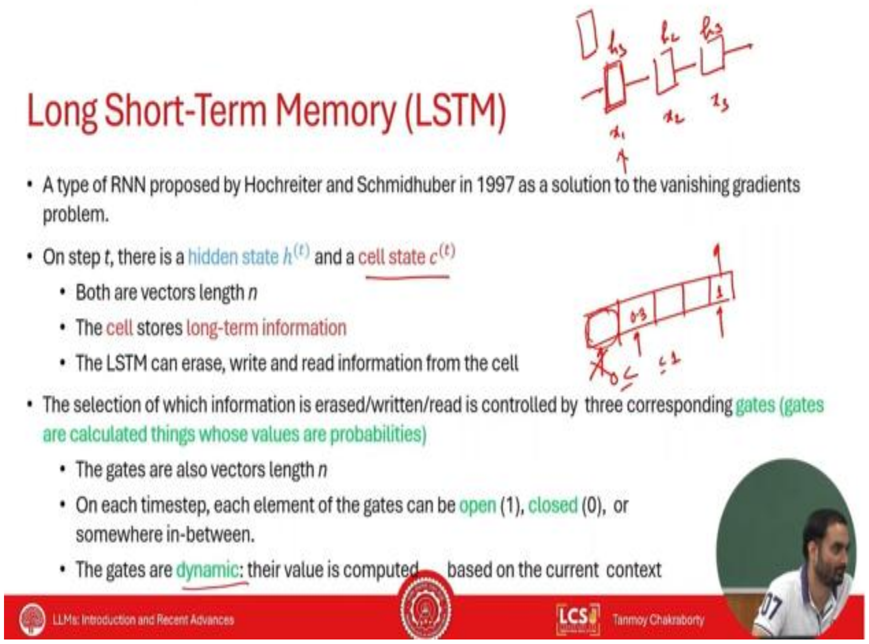
LSTMs & GRUs



Long Short-Term Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
- On step t, there is a hidden state $h^{(t)}$ and a cell state $c^{(t)}$
  - Both are vectors length n
  - The cell stores long-term information
  - The LSTM can erase, write and read information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding gates (gates are calculated things whose values are probabilities)
  - The gates are also vectors length n
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between.
  - The gates are dynamic: their value is computed based on the current context

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty



Long Short-Term Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
- On step t, there is a hidden state $h^{(t)}$ and a cell state $c^{(t)}$
  - Both are vectors length n
  - The cell stores long-term information
  - The LSTM can erase, write and read information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding gates (gates are calculated things whose values are probabilities)
  - The gates are also vectors length n
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between.
  - The gates are dynamic: their value is computed based on the current context

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

So let's focus on LSTM and GRE.

So what is the idea here? So the idea here is that in vanilla RNN, so these are hidden states h1, h2, h3, x1, x2, x3 are the inputs, right. So at every state, at every time you have a hidden state, right. In LSTM we will also maintain another state, a state is a vector, right. We will maintain another state called the cell state, denoted by CT, The cell state will act as a memory. So the cell state is responsible for storing information which you do not want the model to forget, so hidden state and cell state these are the two states which will be available at every time stamp.

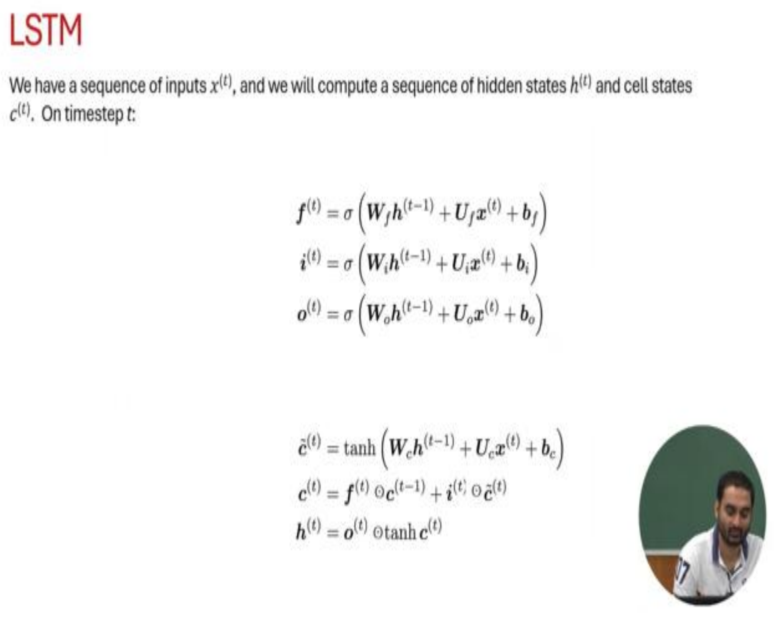
We will control the content which is going to be written to this memory cell state using a getting mechanism. So we have a cell state, we have hidden state, the cell state will act as a memory and the hidden state will act as an output. Now this is the overview. I will discuss in details. We will try to understand the overview of this.

So memory, cell state and hidden state. Cell state will act as a memory and hidden state will act as an output. We will generate some content which we want to store within the memory. Right but how much of this content will be stored into the memory, this will be controlled through a getting mechanism right and what are these gates? These gates are same similar to and or gates that we have that we already know but not as simple as that but the idea is more or less same if the so gate itself is a vector. it is also a vector right.

So, a vector has multiple cells if value of one cell is one, it means it will allow the information to pass, the corresponding information to pass to the memory, if it is 0, it will not allow. If the number is in between 0 to 1 let us say 0.5 or 0.3 proportionately the information will be passed right. In LSTM, we will see there are three kinds of gates that are being maintained.

Now these gates are responsible for deciding which part of the content I would like to store into the memory, how much I would like to read from the memory and store into it instead. So, this operations are very similar to the operations that we have in computer, read and write operation. I write something, so here I write the content to the memory, I read content from the memory and store it to the hidden state. And how much will be stored, how much will be forgotten, how much will be written, all these decisions will be taken by the gates.

Clear. Now these gates the values of these gates are controlled dynamically ok. In fact these are the parameters ok. Values of these gates will be controlled dynamically, gates are vectors like this ok. Values generally the value of every cell of the gate ranges between 0 to 1 If it is 0, then it will not allow the content to move forward. If it is 1, it will allow the complete content to move forward and if let us say this is 0.3, so it will only allow 30% of the content to move forward. Let us look at it.

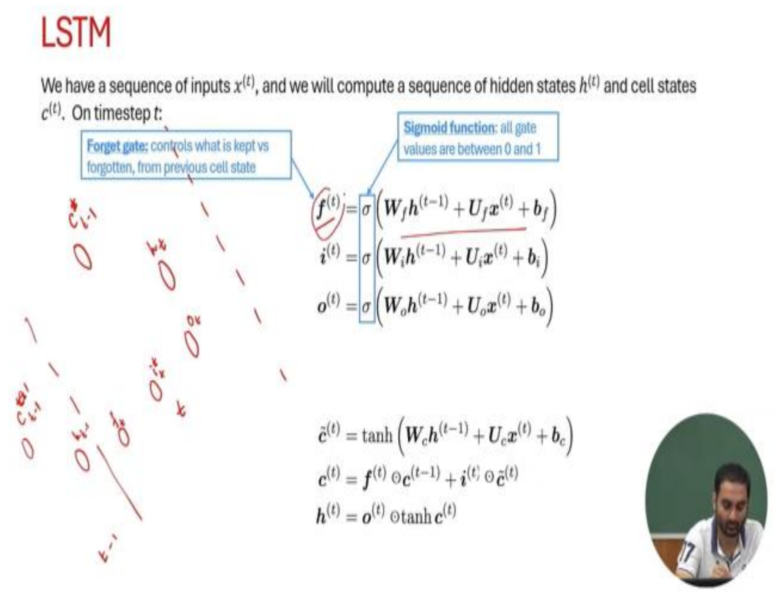


LSTM

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$
$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$
$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)}$$
$$h^{(t)} = o^{(t)} \odot \tanh c^{(t)}$$



LSTM

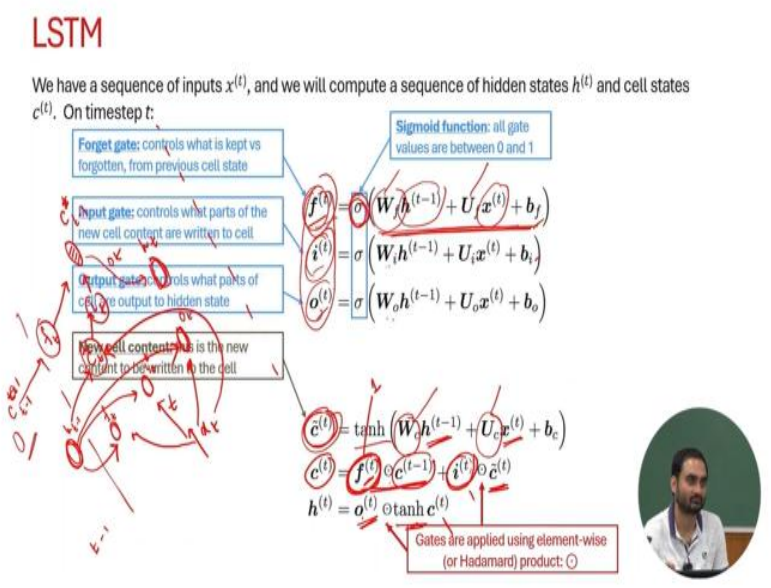
We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Sigmoid function: all gate values are between 0 and 1

$$f^{(t)} = \boxed{\sigma}\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$
$$i^{(t)} = \boxed{\sigma}\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$
$$o^{(t)} = \boxed{\sigma}\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)}$$
$$h^{(t)} = o^{(t)} \odot \tanh c^{(t)}$$



LSTM

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Sigmoid function: all gate values are between 0 and 1

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$
$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$
$$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

$$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)}$$
$$h^{(t)} = o^{(t)} \odot \tanh c^{(t)}$$

Gates are applied using element-wise (or Hadamard) product: ⊙

This is very important. I will go slow so that you understand it. Now as I mentioned at every time t, there is a cell state ct, there is a hidden state ht, this is at every time.

Similarly at t minus 1, we have a cell state c t minus 1, let me write in this way, t minus 1 and a hidden state h t minus 1. So there are three gates that we will create right. One is called the forget gate f t forget gate right. The second one is called the input gate i t and the third one is called the output gate o t. Let me explain the functions of these three gates right.

So, I will discuss these equations later. First try to understand the intuition behind this. So, the forget gate controls what to keep versus what to forget, right. So, now remember at t minus 1 is timestamp, you have some information stored in ct minus 1. Either you want to move the entire information from t minus 1 to t or you want to move a part of this to t right.

So, how much you want to forget and how much you want to move will be decided by f ok. So, you will see that f will be attached with c t minus 1, f will decide how much content right you want to basically move from c t minus 1 to t right. So, what is I t? I t is the input gate. what is the role of the input gate? The role of the input gate is to decide right, so remember at every timestamp you generate some content, t-1 is timestamp you generate some content and how do you generate the content? The content will be generated based on the current input and the previous information right. At this timestamp you generate some content and then we will have to see, that how much of this content that you generate at timestamp t will be written to the cell.

Now remember a cell will have information from the current content and from the previous content. Where was the previous content stored? The previous content was stored in the cell state of previous timestamp. Right, so how much the cell states, previous cell states content will move to the current cell state, who will decide that, forget gate and how much the current content will be stored to the cell, who will decide that, input gate. Clear and then we have, so far there is no role of hidden state, we are just deciding how much amount of information you want to store into the cell state. we have already decided, we have decided cell state has been updated.

Now from cell state how much information you want to read and store it to the hidden state will be decided by the output gate, right. So output gate controls what parts of the cell state are output to the hidden state. Let us look at how these gates are formed. So you see here all these gates F, I, O They are functions of the previous hidden state and the current input. And what are the differences? The differences are the weights.

For the forget gate, you have two sets of weight matrices Wf and Hf. For the input gate Wi, Hi. For output gate Wf, Uf, Wi, Ui, Wo, Uo. Right and this matrices we need to learn and along with this we have bias just ignoring the bias for the time being right. So, all this gates values of this gates are decided by whom? It will be decided by xt right it will be decided by xt and ht-1, okay it will be decided by ht-1, correct, clear, okay. Now what is the next task? The next task is to generate the content at this time, right. So the content that we will generate, this is not the cell content, this is the content that I want to write to the cell, this is called cell content. This is the content that I want to write to the cell. So, the cell content C tilde at time t will be generated based on the previous hidden state same based on the previous hidden state and the current input.

But the look at the parameters, parameters are different WC and UC. So, a cell content C tilde at time t will be generated based on ht minus 1 and xt. Okay now this is the content that I would like to write to the cell state okay but my forget gate and my input gate will not allow to write this totally to the cell state right. So here this forget gate comes to the picture f t and f t will say that look at here Ft and Ct minus 1 right they are connected through this element wise multiplication right. So, Ft will decide how much of the previous current state, previous cell state will be moved forward right.

So, Ft through Ft this will come here and through I_t the cell content that you have just generated will basically come to the memory. So, through IT it will come here. So, the cell state is essentially a function of the previous cell state and the current cell content and it will be controlled by forget gate and input gate. Clear? So based on this we will update the values of the cell content of the cell state. How much I would like to basically write or read from the cell state it will be decided by OT.

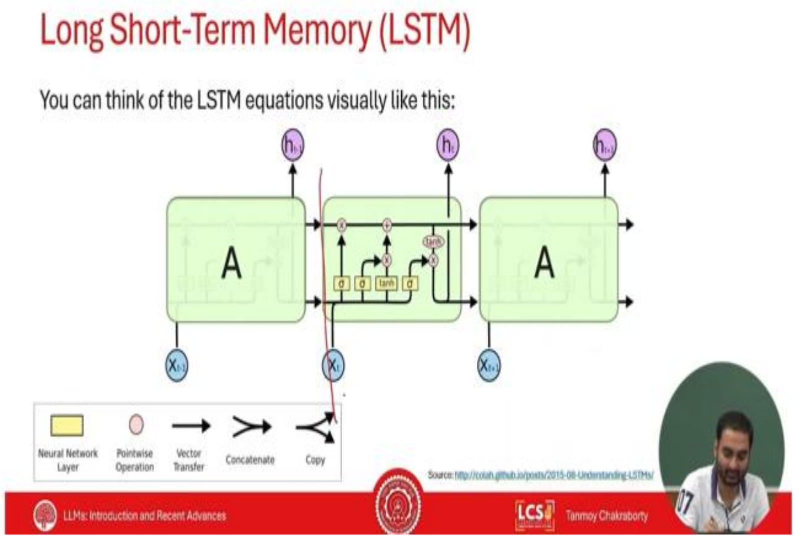
You see here right how much will be how much we would like to read from the cell state and write to the hidden state will be decided by OT. So OT again element wise forget about tan h for the time being right this is OT element wise multiplication CT clear. so now this the current hidden state will be updated. Now think what will happen at t plus 1. At t plus 1 again I will first initialize all the gates right which will be a function of the input at t plus 1, so xt plus 1 and the hidden state ht.

Right and at time t plus 1 we will generate a cell content C tilde right and then the similar operations will basically be executed, okay. Now what is the guarantee that it will not have the vanishing gradient problem? What is the guarantee? or can you intuitively state that you know the past information can be preserved for the longer time. If let us say if f t is 1, all the elements of f t are 1. all the elements of f t are 1 and all the elements of i are 0, then what will happen? So, then this cell state will get copied and copied, right. So, ideally if you do this a token which is far from the current state it will not be forgotten.

But if you want to forget the cell state completely, then you fix f equals to 0 and i equals to 1. So, these two gates will essentially control how much you want to forget, how much you want to retain. Now of course you have seen operations like tan h here, tan h here, these are all design decisions that they took. There are not proper justification behind taking this one, but these are all design decisions.

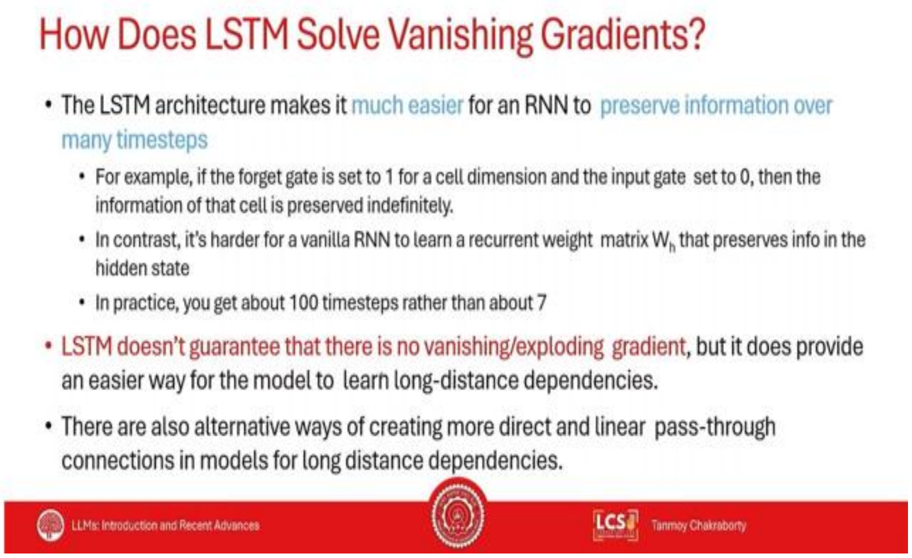
And you have seen that this is passed through a sigmoid. The sigmoid will make sure that ft, it and ot they are just probabilities. Now can you see problems here? How many parameters are there? Now think of a vanilla RNN and an LSTM. LSTM is long short term memory, right? How many parameters are there in vanilla RNN and how many parameters are there in LSTM? So, in LSTM you see if you forget bias for the time being, right, you have two sets of parameters for every get, right and then also you have two sets of parameters for the cell content. There are so many parameters and these are all matrices, remember this, okay. So, LSTM is generally useful when you have a lot of data to train, otherwise you know of it right.

If you have a lot of data to train then only you can use you can basically train all these parameters.


Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:

So this is a diagram pictorial diagram you can I have already mentioned so this is so each of these blocks indicates a particular time stamp right this is one block this is another block another block and so on and so forth.



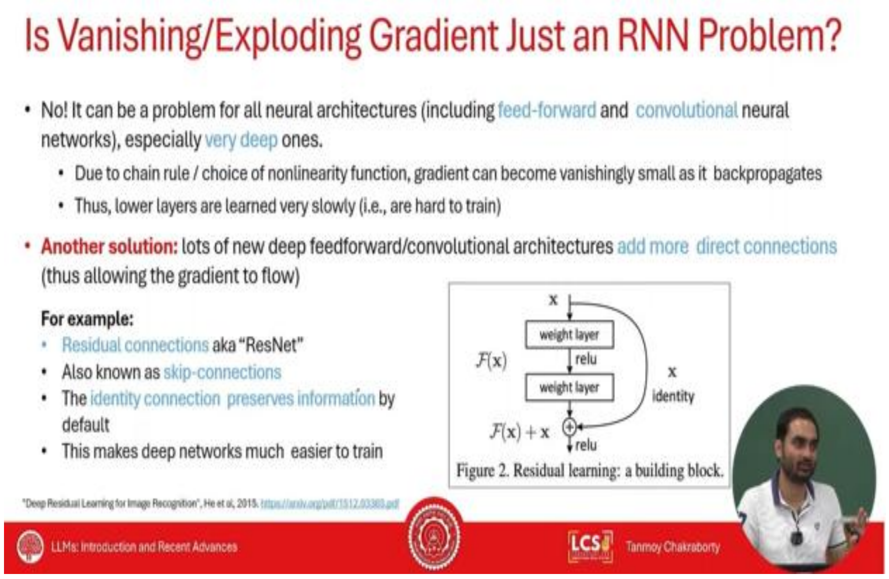
How Does LSTM Solve Vanishing Gradients?

- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
  - For example, if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
  - In practice, you get about 100 timesteps rather than about 7
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.
- There are also alternative ways of creating more direct and linear pass-through connections in models for long distance dependencies.

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

Okay so remember LSTM does not guarantee that it will remove vanishing gradients problem totally, right. But you can say that it will reduce the vanishing gradients problem compared to a vanilla RNA, right. In fact there are other alternative ways of creating more direct and linear path through connections, right.

We will discuss all these things later.



Is Vanishing/Exploding Gradient Just an RNN Problem?

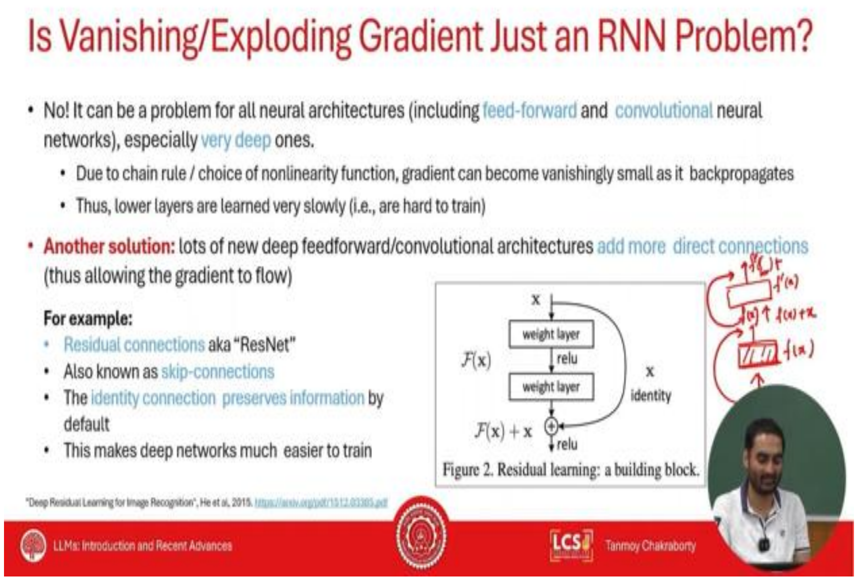
- No! It can be a problem for all neural architectures (including feed-forward and convolutional neural networks), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (i.e., are hard to train)
- Another solution: lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

For example:
- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

Figure 2. Residual learning: a building block.

"Deep Residual Learning for Image Recognition", He et al, 2015. https://arxiv.org/pdf/1512.03385.pdf

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty



Is Vanishing/Exploding Gradient Just an RNN Problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional neural networks), especially very deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus, lower layers are learned very slowly (i.e., are hard to train)
- **Another solution:** lots of new deep feedforward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

For example:

- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

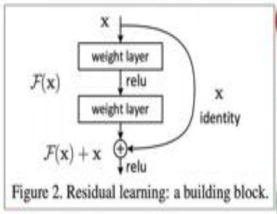


Figure 2. Residual learning: a building block.

"Deep Residual Learning for Image Recognition", He et al, 2015. https://arxiv.org/pdf/1512.03385.pdf

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

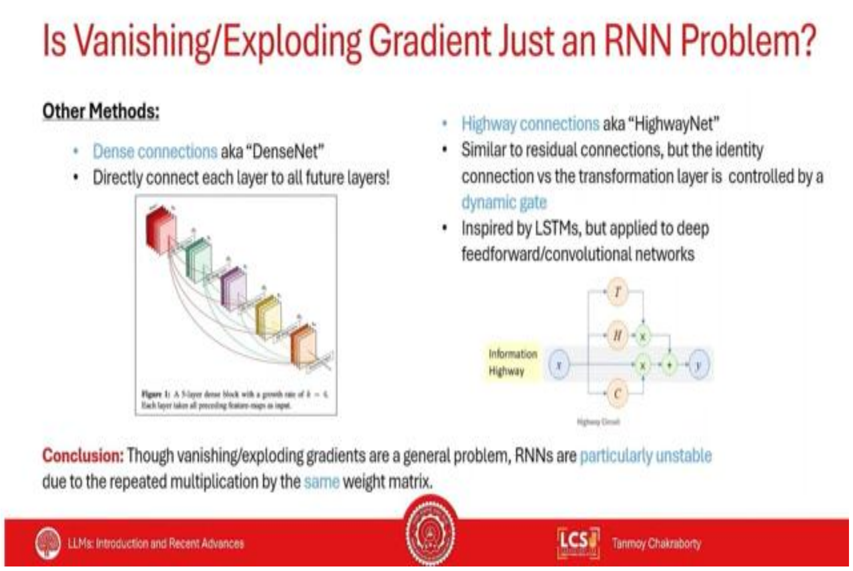
Okay so one way, so LSTM was one way, the other way to address vanishing gradient is what? Now remember the vanishing gradient problem is not an unique problem for LSTM, right. Vanishing gradient problem is there in other models also. For example if you think of a very deep feed forward network. like very deep feed forward network. If you are ever a transformer, for example, you know that transformer has multiple layers, 12 layers, 24 layers, 36 layers, and so on and so forth.

So as we increase the number of layers, you see very significant problem, right. So as we increase the number of players in simple CNN, simple RNN, simple feed forward network, you see that the effect of the bottom layers to the loss, final loss will be much lesser. So you see the bottom layers will not be updated at all. It will remain the same over the iterations, right. So another way to address this problem is something called a residual connection.

This is also called skip connection. What is the residual connection? A residual connection says that let's say you have a layer like this, okay. A layer is nothing but a function, isn't it? It's a function, right. Let's say this is your x, right, the input to that layer and the layer output is f of x, right and let's say you want to pass this f of x again to another layer which is f dash x. right and so on. So what you do in this residual connection you essentially you add the input to the output.

So the input to the next layer will be fx plus x, okay. Again when you generated f dash something right you feed this input here right you basically take a plus of this. Now what will happen is that due to so many complicated operations within each layer, the model will start forgetting the input itself. So every time you basically make it awake that please don't remember the input. You have an input and you should not forget that input.

So at every step you keep on adding this input again and again. We will see this kind of skip connection in transformer when we discuss later.



Is Vanishing/Exploding Gradient Just an RNN Problem?

Other Methods:

- Dense connections aka "DenseNet"
- Directly connect each layer to all future layers!

Figure 1: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input.

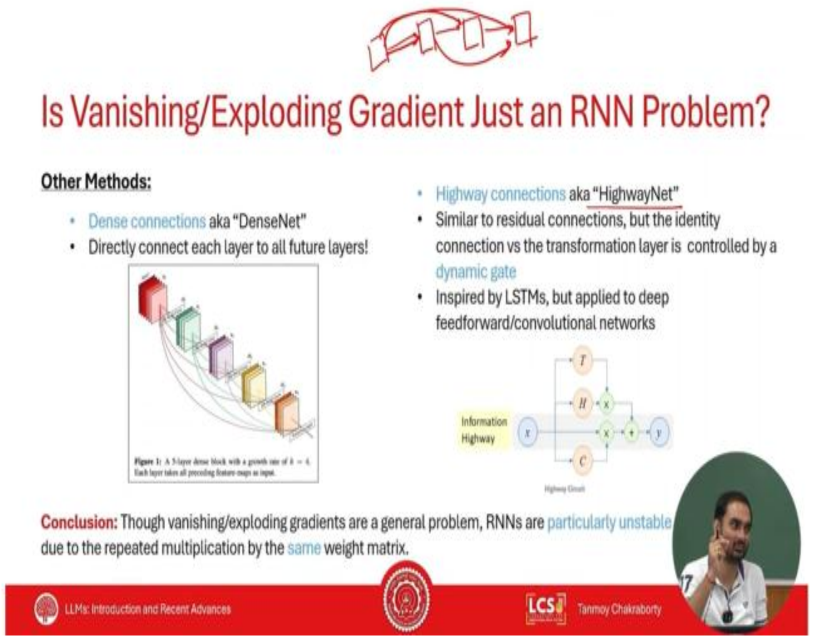
- Highway connections aka "HighwayNet"
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks

Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix.

LLMs: Introduction and Recent Advances

LCS

Tanmoy Chakraborty



Is Vanishing/Exploding Gradient Just an RNN Problem?

Other Methods:

- Dense connections aka "DenseNet"
- Directly connect each layer to all future layers!

Figure 1: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input.

- Highway connections aka "HighwayNet"
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks

Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix.

LLMs: Introduction and Recent Advances

LCS

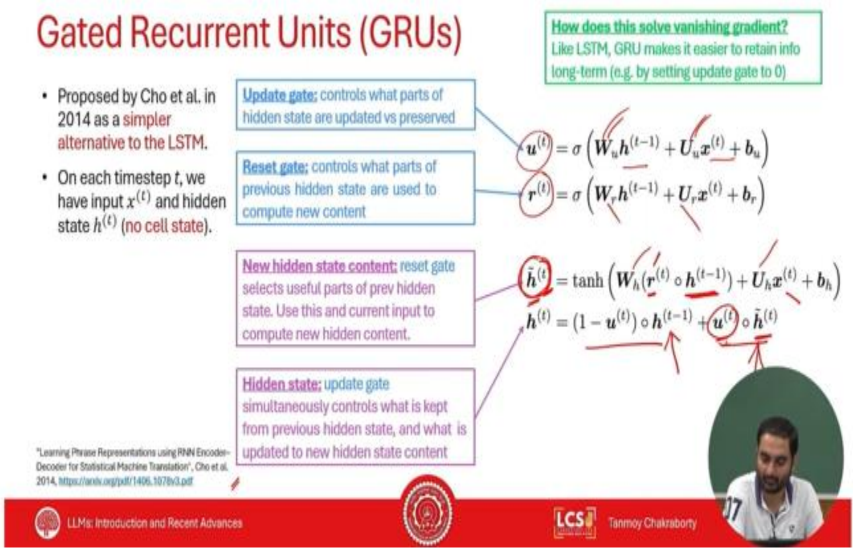
Tanmoy Chakraborty

The other way to address this is let's say a dense connection, dense net. In dense net what happens? Let's say you have RNN layers like this. So in a vanilla RNN, two consecutive layers are connected through one connection. In dense net, what you do, you connect all the layers to all its next layers, each of the layers to all its next layers.

For example, there is a connection from here to here, there is a connection from here to here, there is a connection from here to here, from here to here and so on and so forth. So every layer is connected to all the future layers. Not back connection by the way, forward connection. Okay. So in that way you will always make sure that information will not be forgotten because you have multiple connections.

Of course how to combine all these connections together would be a question but there are ways. In fact, there are many other ways. There is another technique called highway connection where you can use a gating mechanism. So highway connection is a mixture of a gating mechanism and dense net.

So this is called highway net. In highway net, you use a gate to control So, you also have this dense connections and this gate. So, gate will control how much you want to basically remember from the far-far away states and how much you want to remember from the current state, okay, the combination of this.



Gated Recurrent Units (GRUs)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep t, we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

How does this solve vanishing gradient? Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h (\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

"Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

LLMs: Introduction and Recent Advances

LCS2 Tanmoy Chakraborty

Okay, then, so LSTM was proposed long time back by the way. long time back and people did not use it because at that time, let's say early 2000, that time there was a scarcity of the amount of data. And then in 2014, so from 2008, 2009 when people started curating data, huge amount of data.

Dale's net, Alex's net, all these methods were proposed in 2011, 2012. People got access to huge amount of data sets. Then they thought that you know LSTM is okay, but let's look at other mechanisms. So in this paper in 2014, right, they proposed a mechanism called GRU, Gated Recurrent Unit. So in GRU the idea is exactly the same as LSTM. The only difference is that here there is no explicit cell state and there are only two gates.

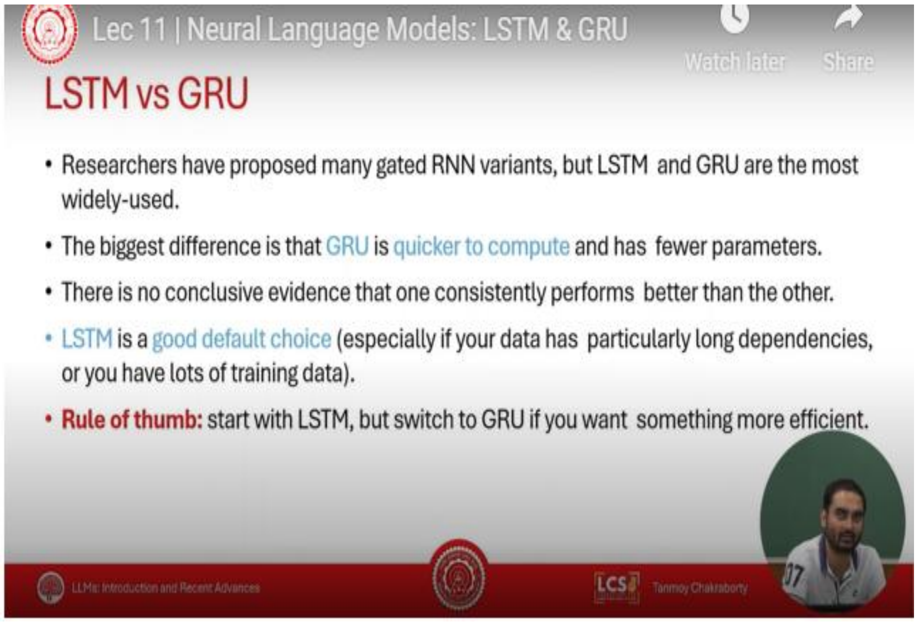
One is called the update gate which is responsible for controlling what parts of the hidden state are updated versus what to preserve. So there's only one state which is hidden state. You have to control. So this is what you have. So you want to control how much of the previous hidden state will be preserved and how much of the previous hidden state will be updated in the next hidden state.

A reset gate controls what parts of the previous hidden states are used to compute the new content. Let's look at the equation. So similar to LSTM here also I will create I will basically initialize these two vectors create these two vectors gates update gates and reset gate and their functions of the previous hidden state and the current same right with parameters different parameters. Now this time, here there is no cell content. Here there is something called hidden cell content that you want to insert into hidden state.

Now this hidden state content H tilde is essentially controlled by R. What is R? R is a reset gate. So R will decide how much of the previous hidden states will be used to generate the content okay and then so you generated the content which is H tilde right which is a function of the previous hidden state and the current input right and then this content will again be controlled by the update gate. So the update gate will say decide So, there are two components now, one is the current hidden state content that you generated H tilde and the previous hidden state, right. Then this, the update get controls that how much of the current cell, current content will be moved to the hidden state, current hidden state and how much of the previous hidden state will be moved to the current hidden state, right.

And you see they use a single gate ut, right. So ut and here ut and here 1 minus ut, right. So it basically balances. If ut is very high then we will give low weightage to the previous hidden state and more weightage to the current content if ut is low and then you basically do the opposite. Okay so in this way we essentially control the entire thing.

Now you see how many parameters are here. There are two parameter sets, two parameter matrices for U, there are two parameter matrices for R and there are two parameter matrices for H tilde. Okay.


Lec 11 | Neural Language Models: LSTM & GRU
Watch later
Share
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

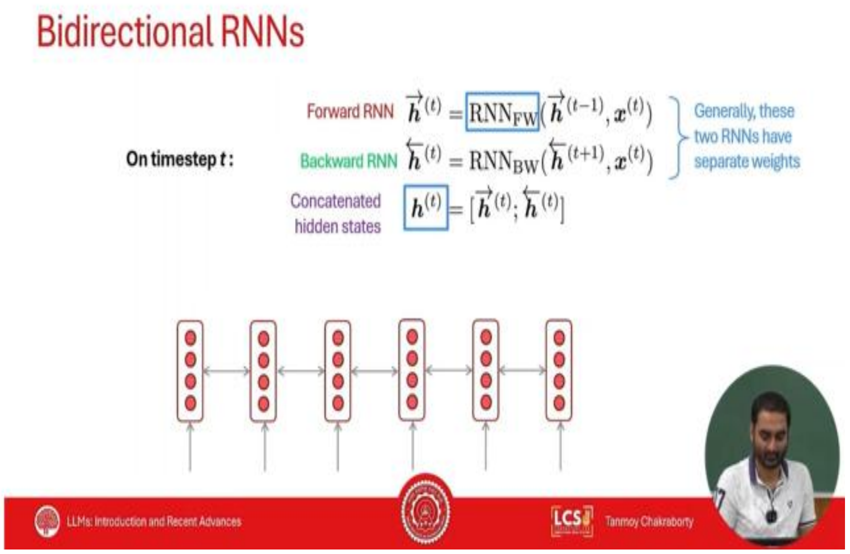
LSTM vs GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used.
- The biggest difference is that GRU is quicker to compute and has fewer parameters.
- There is no conclusive evidence that one consistently performs better than the other.
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data).
- **Rule of thumb:** start with LSTM, but switch to GRU if you want something more efficient.

Let's look at the comparison between LSTM and GRU. You can ask that which one is preferred, LSTM or GRU. In general, the suggestion is that you start with LSTM and then you move to GRU if you have enough data.

If you do not have enough data, you can use GRU. But this is the rule of thumb. You start with LSTM, you can switch it to GRU if you want something more efficient. But there is no clear evidence which one is better, LSTM or GRU. Sometimes LSTM performs better, sometimes GRU performs better.

Okay, so this is about LSTM GRU.



Bidirectional RNNs

On timestep t:

Forward RNN $\overrightarrow{h}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{h}^{(t-1)}, x^{(t)})$

Backward RNN $\overleftarrow{h}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{h}^{(t+1)}, x^{(t)})$

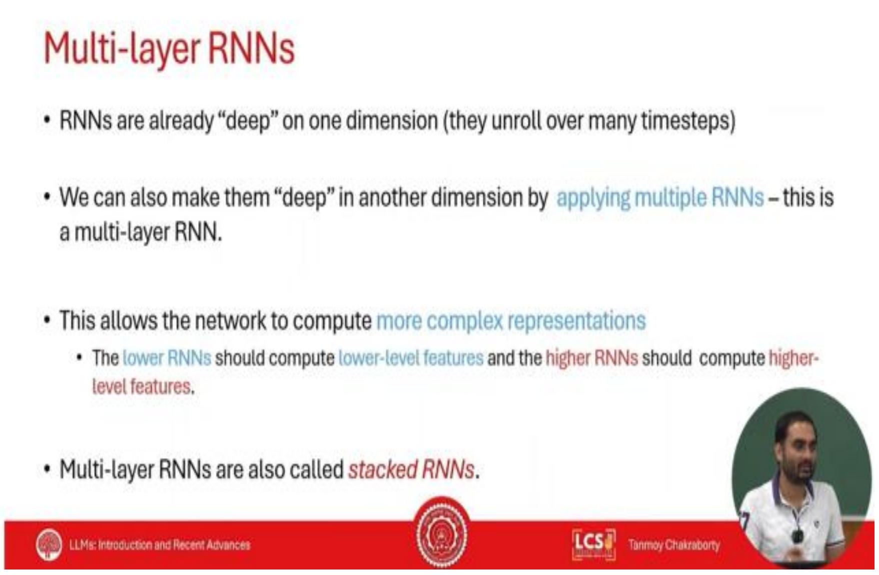
Generally, these two RNNs have separate weights

Concatenated hidden states $\boxed{h^{(t)}} = [\overrightarrow{h}^{(t)}; \overleftarrow{h}^{(t)}]$

LLMs: Introduction and Recent Advances

LCS2 Tanmoy Chakraborty

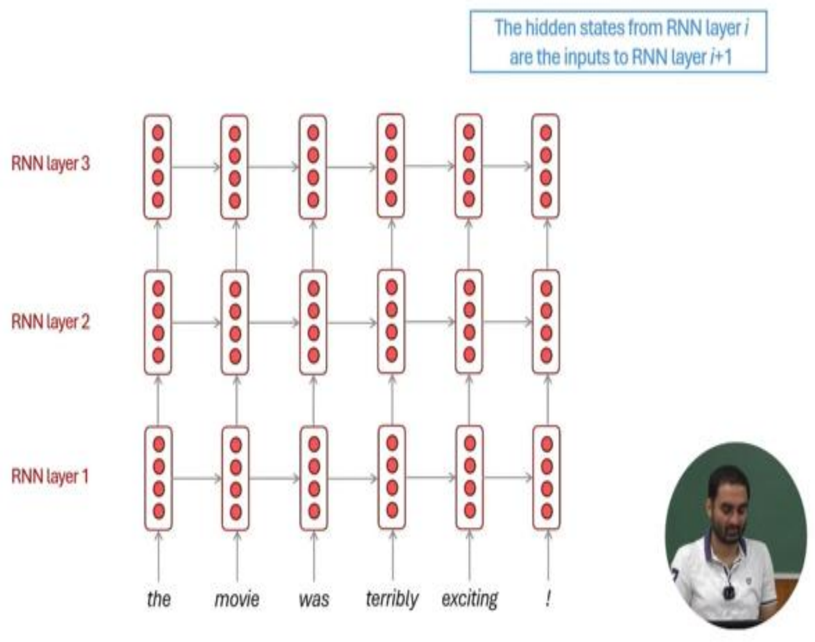
I already mentioned bidirectional RNN right. In bidirectional RNN you see this bidirectional arrow right, so there is one forward RNN right which will generate hidden state at t, there is a backward RNN right which will generate a hidden state at t and then the final hidden state at t is basically a concatenation of the backward RNN hidden state and forward RNN hidden state. Right and this kind of methods you use when you want to access to the entire sequence, right.



Multi-layer RNNs

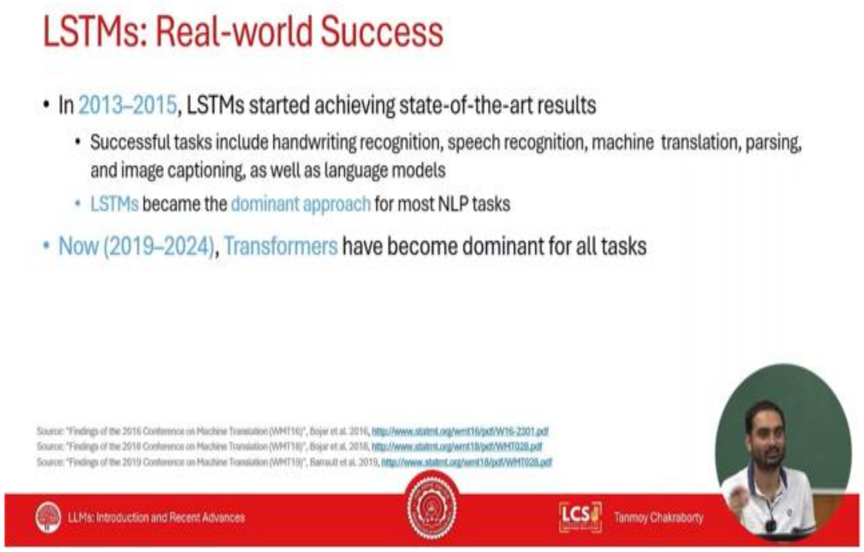
- RNNs are already "deep" on one dimension (they unroll over many timesteps)
- We can also make them "deep" in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty


The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1
RNN layer 3
RNN layer 2
RNN layer 1
the
movie
was
terribly
exciting
!

Multi-layer RNN, you can also think of other variations where RNN layers are stacked one upon another, right. So this is one RNN, it can be bidirectional, unidirectional, this is another RNN but look at here, what's the difference between bidirectional and multi layer? In bi-directional you have something like this. So this is a time stamp T okay where this hidden state will be generated based on the current input Xt and the previous hidden state Ht plus 1 right and Xt.

This hidden state will be generated based on Xt and Ht minus 1. Right and these two things will be combined but in multi-layered RNN, you see here the input of every hidden state is not the actual input. What is the input? The input is the output of the previous hidden state, output of the hidden layer of the previous, hidden state of the previous layer. See the difference between these two.



LSTMs: Real-world Success

- In 2013–2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks
- Now (2019–2024), Transformers have become dominant for all tasks

Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf
Source: "Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, http://www.statmt.org/wmt18/pdf/WMT028.pdf

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

LSTM has been very very successful from 2013 to 2015 and then in 2017 transformer was proposed and then people started using transformer.

But again the importance of LSTM kind of methods was realized recently. The state space model, Mamba kind of model, right, we will see that their RNN models will be used, right. RNN has its own beauty, right. But here also the same, I mean one problem still remains there. What's the problem? You cannot process the sequence in parallel.

You can't process the sequence in parallel, right. You have to wait till t minus 1 to basically process the hidden state at t, right. We will see how we can address this, okay. Thank you.