Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

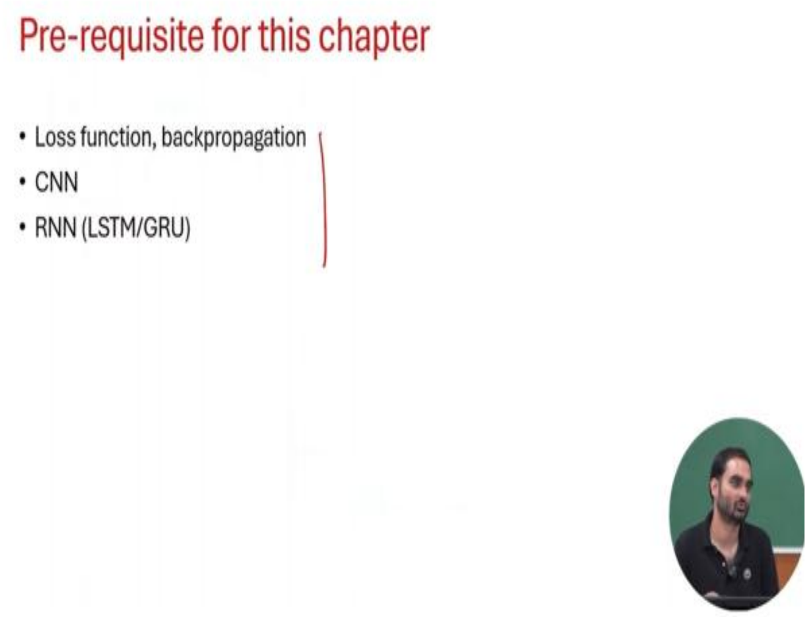
Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

Lecture 10

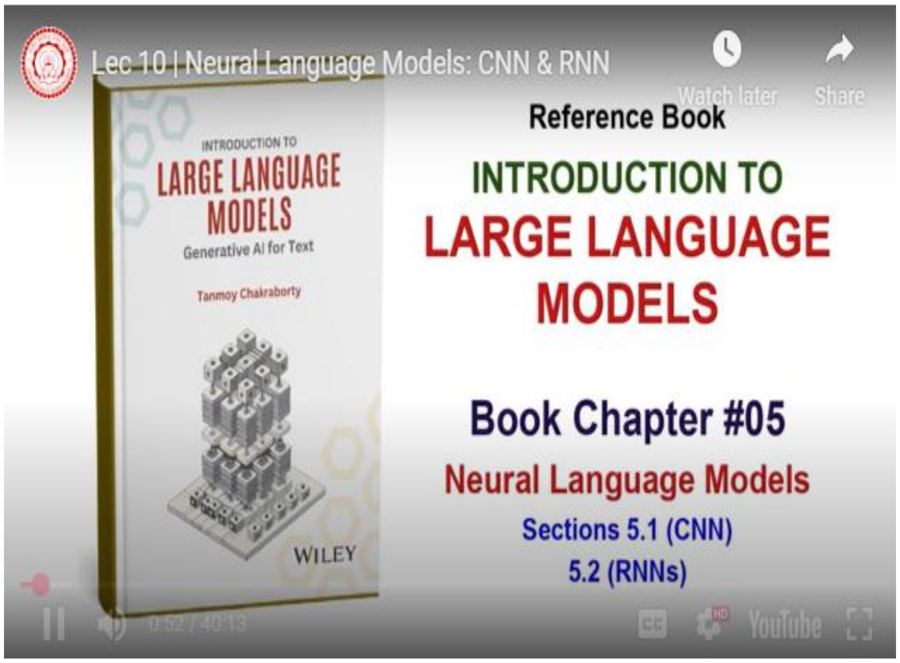
Neural Language Models: CNN & RNN

Hello everyone, so in this lecture we will start discussing neural language model. So in the last class we have completed word embedding where we talked about glove and before that we also discussed word to wake and count based methods. We will see how these methods are used in neural language model.



Pre-requisite for this chapter

- Loss function, backpropagation
- CNN
- RNN (LSTM/GRU)


Lec 10 | Neural Language Models: CNN & RNN

Reference Book

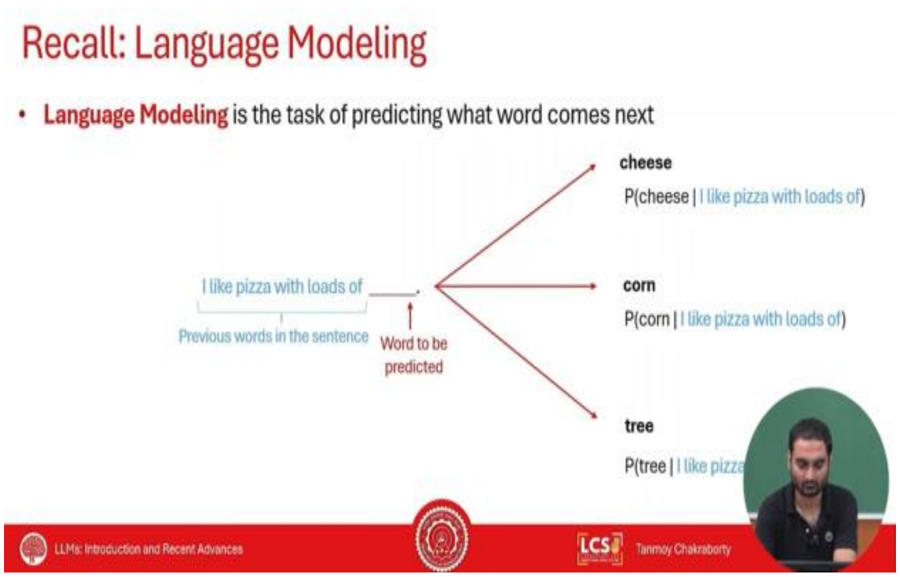
INTRODUCTION TO LARGE LANGUAGE MODELS

Book Chapter #05

Neural Language Models

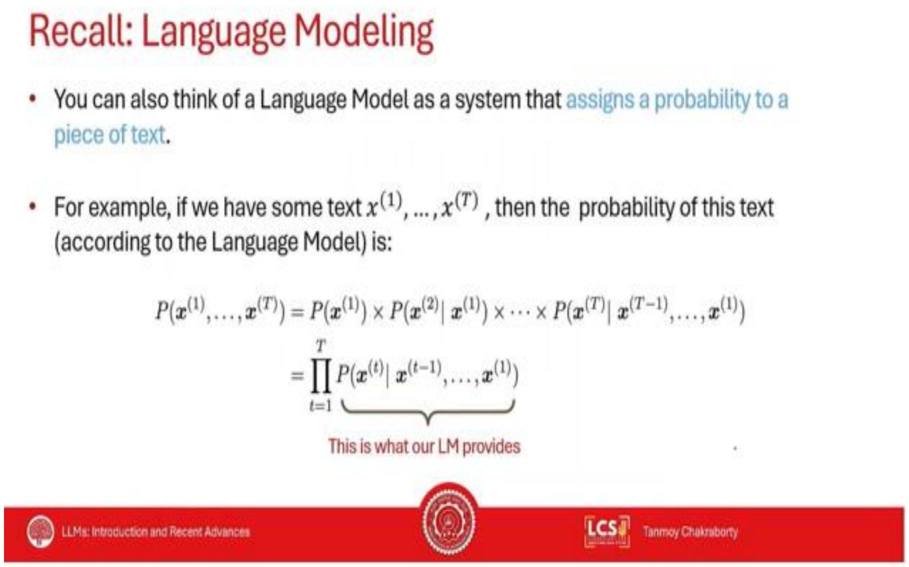
Sections 5.1 (CNN)
5.2 (RNNs)

Okay, for this chapter, these are the prerequisites: you should know what a loss function. how back propagation happens right, you should have a fair bit of understanding about CNN and little bit of understanding about RNN also okay.


Recall: Language Modeling

- **Language Modeling** is the task of predicting what word comes next

I like pizza with loads of ____.
Previous words in the sentence
Word to be predicted
cheese
P(cheese | I like pizza with loads of)
corn
P(corn | I like pizza with loads of)
tree
P(tree | I like pizza
LLMs: Introduction and Recent Advances
Tanmoy Chakraborty



Recall: Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text.
- For example, if we have some text $x^{(1)}, \dots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)}|\,\boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)}|\,\boldsymbol{x}^{(T-1)},\dots,\boldsymbol{x}^{(1)})$$
$$= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)}|\,\boldsymbol{x}^{(t-1)},\dots,\boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}}$$

LLMs: Introduction and Recent Advances

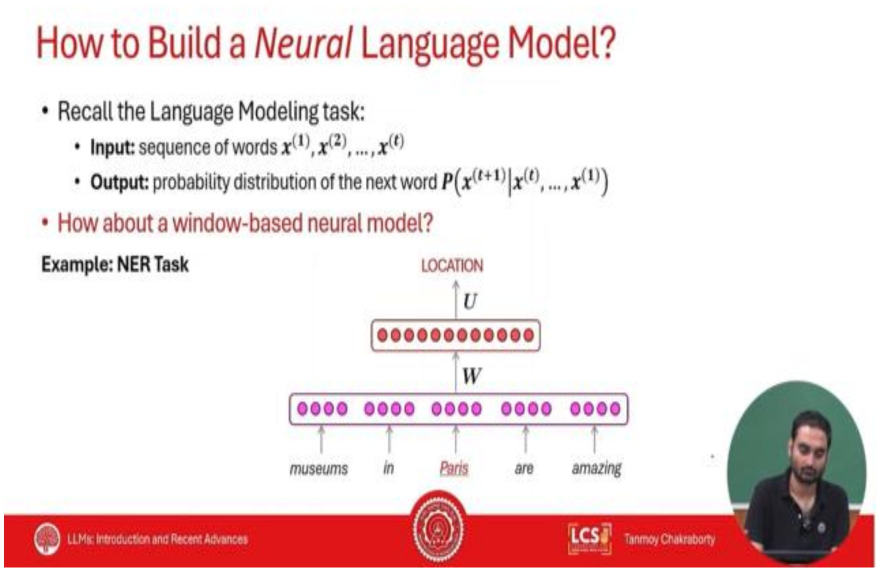
LCS2

Tanmoy Chakraborty

Okay so let's get started, this is a neural language model.

We already discussed statistical language model. If you remember, the task was to predict the next word. This was the example that was shown. So I like pizza with lots of dash.

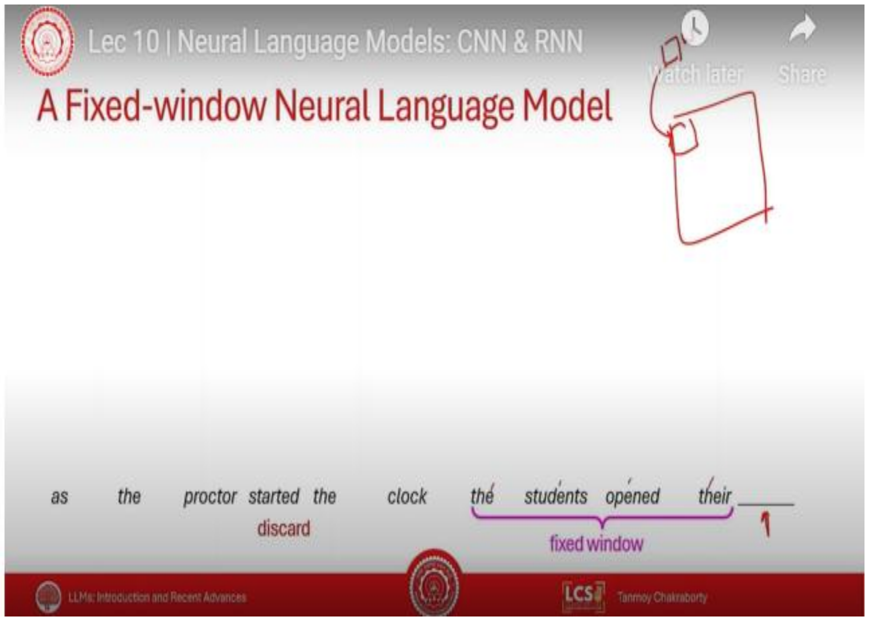
And the language model should be able to predict the appropriate word. And how the language model predicts? It basically samples a word from a distribution, what is the distribution? It depends on the hypothesis. Let's say if you are designing a unigram language model, so distribution is sampled from a unigram model, otherwise bigram or trigram and so on and so forth. What are the major problems in statistical language model? The major problem is that for an unknown word, it would be very difficult to process it. The bigram matrix that you compute would also be sparse and so on and so forth.



How to Build a *Neural* Language Model?

- Recall the Language Modeling task:
  - **Input:** sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
  - **Output:** probability distribution of the next word $P\left(x^{(t+1)} \middle| x^{(t)}, \dots, x^{(1)}\right)$
- How about a window-based neural model?

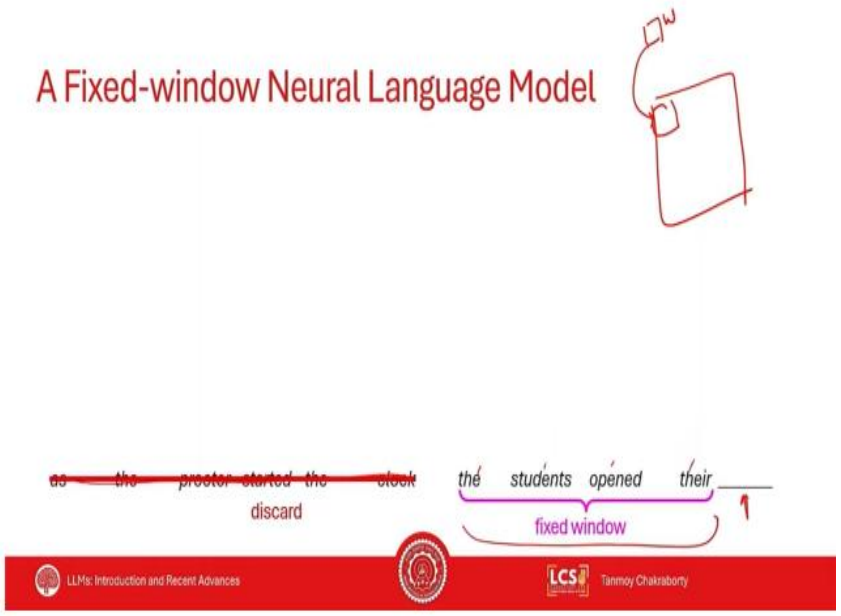
Example: NER Task



Lec 10 | Neural Language Models: CNN & RNN

A Fixed-window Neural Language Model

Watch later
Share
as the proctor started the clock the students opened their ____
discard
fixed window



A Fixed-window Neural Language Model

W
~~as the proctor started the clock~~
discard
the students opened their ______
fixed window
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

So now we move to a neural language model. It was proposed by Yoshua Bengio in 2001, 2002 during that time. Not exactly the one that I am describing here but you know somewhat similar to what we will discuss here. Okay. So we will start with the CNN method and we will see that how a CNN method, a CNN model predicts the next word.

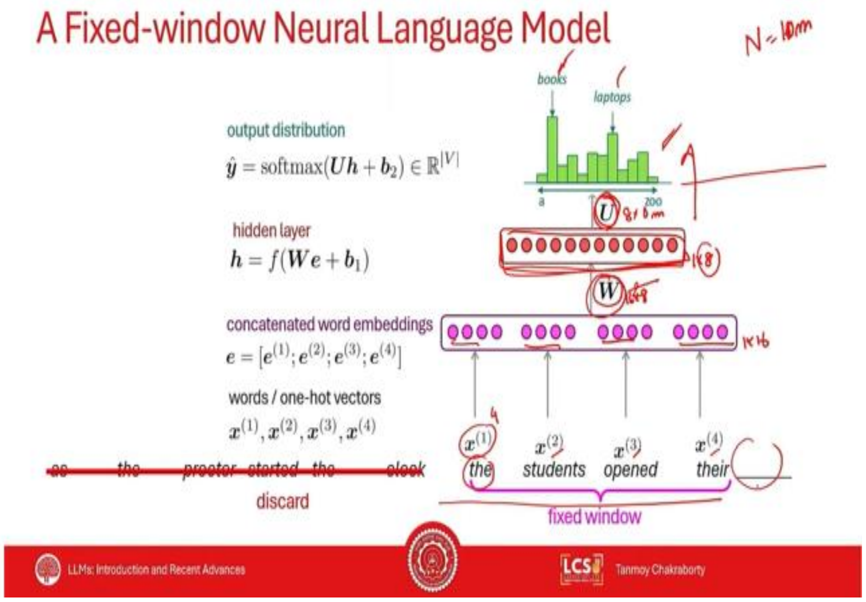
Okay. For a CNN method, we all know that when we run a CNN on an image, right, let's say an RGB kind of image, what we do? We identify a particular patch, right? Let's say this is an image, right? We first identify a particular patch and on top of this patch, we apply a kernel or a filter, right? What is this kernel or filter? This is essentially a matrix W. which is same as the size of this patch, right. We apply this here and then we can use let's say max pooling, min pooling, etcetera, etcetera, right. These are, I mean you can also think of some non-linear function, right, out of whatever you got, right, from this kernel.

So you need to first identify a patch size, right, a window size. Here, in our case, in case of language or a sentence, a sentence is a sequence of tokens, right? So I also need to identify a context or whatever, a path size here also. It's a 1D matrix. It's not a 2D matrix. It's a 1D matrix where there's a row, and in that row, the tokens are arranged, right? And these tokens are essentially represented by what I'm getting.

So to use a CNN, what we need? We need to first identify a window size or a filter size, a kernel size. We can't process the entire sequence at one go. So let's say this is the sentence.

As the proctor started the clock, the students opened their dash. and you want to identify the next word, right.

And let's say the size of this window is 4, the size of this kernel is 4, right. So you look at the previous 4 words and you discard all these other words, okay. So you discard them, okay. So this is now the patch that we are focusing on, okay.


A Fixed-window Neural Language Model
N = 10m
books
laptops
output distribution
$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$
a
zoo
U
hidden layer
$h = f(We + b_1)$
1×8
W
concatenated word embeddings
$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$
1×16
words / one-hot vectors
$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$
~~as the proctor started the clock~~
discard
the
students
opened
their
fixed window
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

Now we apply a CNN kind of layer on top of this.

So the is represented by this vector x1, this is x2, x3 and x4. x1, x2, x3, x4 are word embedding vectors obtained from let's say word2vec or glove or it can be as simple as one-hot encoding. Then let's say the size of the vector is 4. So 4, 4, 4 and 4. So it's a 1 cross 16 matrix.

And on top of this I apply the kernel or a filter W. W is a matrix. When I apply this, let's say the size of this matrix is 16 x 8 for example. When you multiply this, it will produce a vector size of 1 x 8. whatever 18 you know transpose you just look at the dimension okay so it will produce another vector of size 8 yes or no right so now this is my hidden state.

What is the goal? The goal is to predict what is going to be the next word. And how many words are possible? How many candidates words are possible? Vocabulary size. Let's say there are n number of, let's say n is 1 million. So there are 1 million number of words present. Any of these words can basically come here.

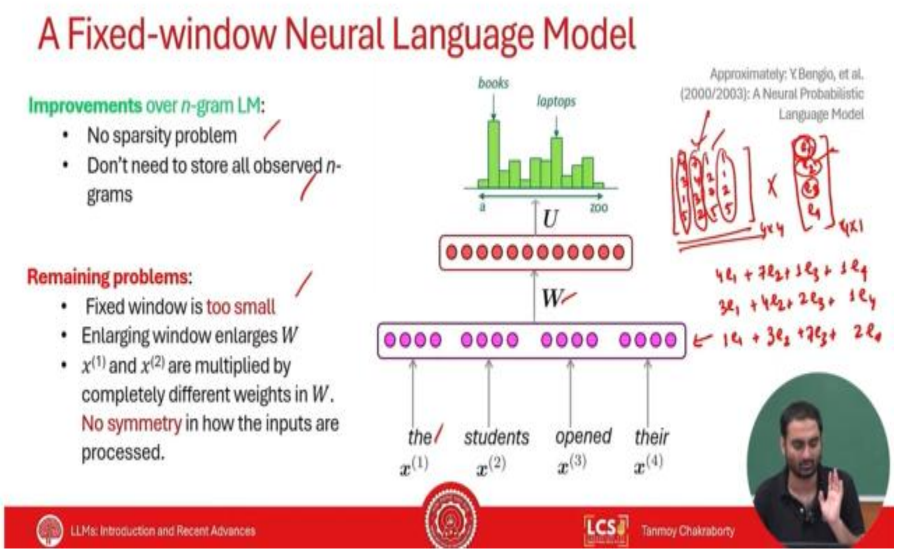
So from this hidden layer right what I do? I apply another linear layer. Linear layer is just a feed forward layer right which will basically up project right this 8 size vector to a million size vector right. So what is the size of, what is the dimension of this u 8 cross 1 million. Its just an example by the way, okay. So it will produce a vector of size 1 million and of course after this projection you apply short max and all this stuff so that the entries of this vector will be basically probabilities and all the probability laws will be followed, okay.

So I will get the distribution like this. From that distribution, I will sample. And hope is that the word with maximum probability will be sampled. So this word can be either books or laptops, for example.

This is CNN. Clear? So you flatten the entire matrix. You apply CNN layers. You can have multiple such layers, by the way. There's only one layer I showed. You can have multiple such layers.

OK? and then you sample it. What is the problem in this kind of approaches? The first problem is that we have to decide the window size, right? We can't consider the entire context as window. The second problem is that the size of the window, which is four in this case, is an hyperparameter, meaning you have to play around this. You have to play with multiple values of this window, and you have to see which one is better. So as we increase the size of the window, the dimension of W will change.

It will also increase. So with the increasing size of context, with the increasing size of window, the dimension of the weight matrix that we are going to learn will also increase, right? These are the two major problems. But what are the good part about the CNN method over statistical language model? What do you think? Do you think the sparsity can be addressed here? There will be no zero, right? It can also handle unknown words in a very systematic way, right? Assuming that the word embeddings are available, otherwise you can use word not encoding and so on.

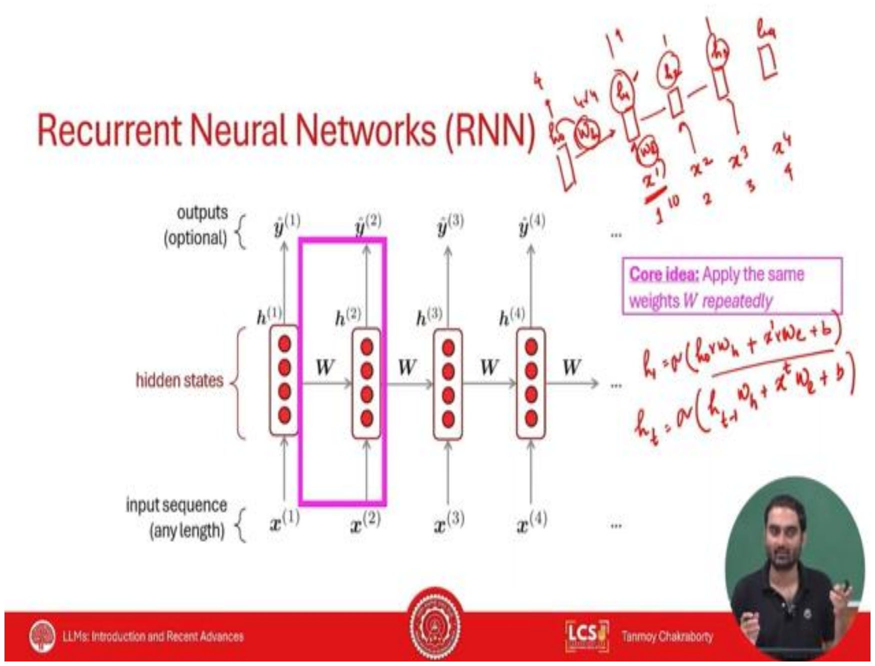


A Fixed-window Neural Language Model

Improvements over *n*-gram LM:
- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining problems:
- Fixed window is too small
- Enlarging window enlarges W
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model



Recurrent Neural Networks (RNN)

Core idea: Apply the same weights W *repeatedly*

$$h_1 = \sigma(h_0 W_h + x^1 W_e + b)$$

$$h_t = \sigma(h_{t-1} W_h + x^t W_e + b)$$

So these are the good parts. You don't need to store this entire matrix by the way. You don't need to store the entire bigram matrix here.

Okay. The other problem, the problem that I mentioned, the first problem is that you need to decide the size of the window, right? With the increasing window size, W will increase. The third problem is more subtle. Let us discuss the third problem. What is this third problem? The third problem is that, let's say, you have this W matrix right which is a 4 cross 4 matrix let us say 4 cross 4 right. Let us say 4315 7432 1275 1125 this is the W matrix okay and let us say this is my embedding, this is the embedding matrix which will be multiplied with W right and let us say the embedding matrix is like this e1, e2, e3, e4.

Let us say E1 is the embedding for the word the and E2 is the embedding for the word students and so on and so forth. This is a 4 cross 1. You have to multiply these two. Right and let us assume that these embeddings are one dimensional embeddings just for the sake of discussion okay. If you multiply what is going to happen 4E1 plus 7E2 plus 1E3 plus 1E4 right 3E1 plus 4E2 plus 2E3 plus 1E4 right.

1 E1 plus 3 E2 plus 7 E3 plus 2 E4. So what is actually happening? You notice what is happening here? E1 is only affecting this column. E2 is only affecting this column. E3 is affecting this column and E4 is affecting this column. So ideally when we do this kind of multiplication, weight and embeddings and so on and so forth, we hope that you know, it should not be disjoint something like this, right? The hope is that when we multiply the information from U1 should pass through all the columns of weights.

The information of W2 should, E2 should also pass through all the columns of the weights and so on and so forth. I mean, when I multiply something with something, what I'm doing essentially, I'm essentially doing some sort of message passing. But here if you look at it carefully E1 is not affecting column 2, E2 is not affecting column 3 and column 1 and so on and so forth. So essentially the message passing is not actually happening at all. You are essentially learning four independent functions.

It is a very subtle, think about it carefully. It is not very easy to understand. You pause and ponder and then maybe we will discuss again. So there are three problems. The first problem is that it depends on the fixed window size.

It is not able to capture infinite length window. The second problem is that there is no message passing happening. And the third problem is that with the increasing size of window, W is also increasing. So let us address this using RNN. What is an RNN? So an RNN looks like this.

So let's say I have inputs x1, x2, x3, and x4. These are embeddings. These are embeddings. And we have different hidden states. So at every position, we have a hidden state.

So this is position 1, position 2, position 3, position 4. Then also think of these positions as times. Time 1, time 2, time 3, time 4. And this is hidden state 1, hidden state 2, hidden state 3, hidden state 4. And let's say there is a dummy hidden state H0.

Let's start with. Okay. And how we are computing every hidden state? Every hidden state is computed in this way. Let's say H1 is computed. So there is a weight parameter matrix WH here. Right there is a, so this parameter matrix is essentially a projection a parameter matrix right and you have another parameter matrix we for example here. So h0 will be multiplied with wh right x1 will be multiplied with we this will be added of course some biased term and then I will pass it through a non-linearity this will be multiplied with this, this will be multiplied with this, we will add them, add a bias and we will basically pass it through some non-linearity, sigma y dot tan h, that will give you the hidden state.

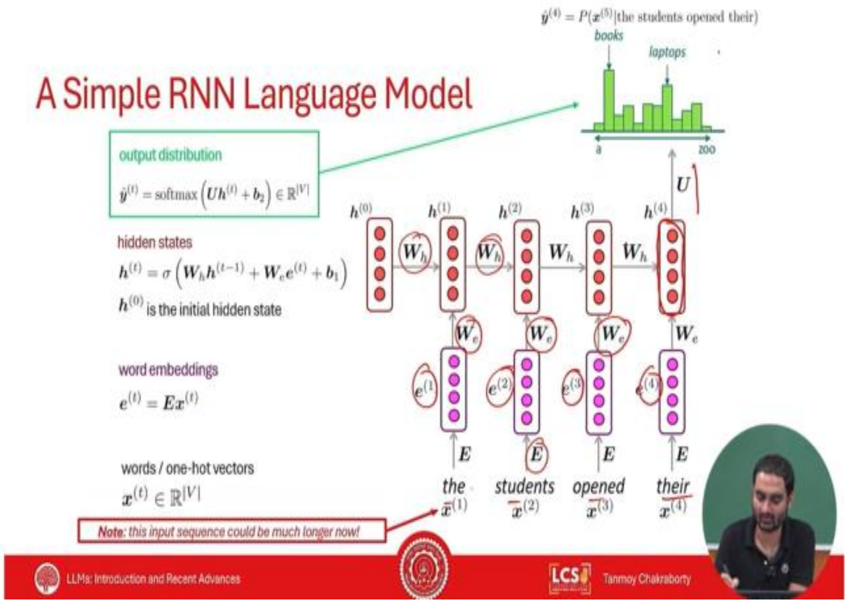
So now look at here, h1 is determined by h0 and x1, h2 is determined by H1 and X2, H3 is determined by H2 and X3 and so on and so forth. So essentially HT is determined by HT minus 1 WH plus XT WE plus a bias B right. If you look at it very carefully this is a recurrent function. Right and these two weight matrices WH and WE they are the same across all the layers so here there are four layers right so they are the same across all the layers and these matrix WH and WE they are independent of the context size. There is no concept of context at all but if you still want to you know consider the context it is independent of the size of the context.

$$h_1 = a(h_0 r w_h + x' r w_e + b)$$

$$h_t = a(h_{t-1} w_h + x^t w_e + b)$$

What is the dimension of WH by the way? So let's say the dimension of the hidden state is 4. It's a vector right and this is also 4. So the dimension of WH will be 4 cross 4. What is the dimension of WE? Let us say the dimension of the embedding is 10 and the dimension of hidden state is 4. So this is 10 cross 4 or whatever 4 cross 10 and this is fixed irrespective of the size of the context length.

It only depends on two things. One is the size of the hidden state that we will define beforehand and the size of the embedding that is also constant.



A Simple RNN Language Model

output distribution

$\hat{y}^{(t)} = \text{softmax}\left(Uh^{(t)} + b_2\right) \in \mathbb{R}^{|V|}$

hidden states

$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$

$h^{(0)}$ is the initial hidden state

word embeddings

$e^{(t)} = Ex^{(t)}$

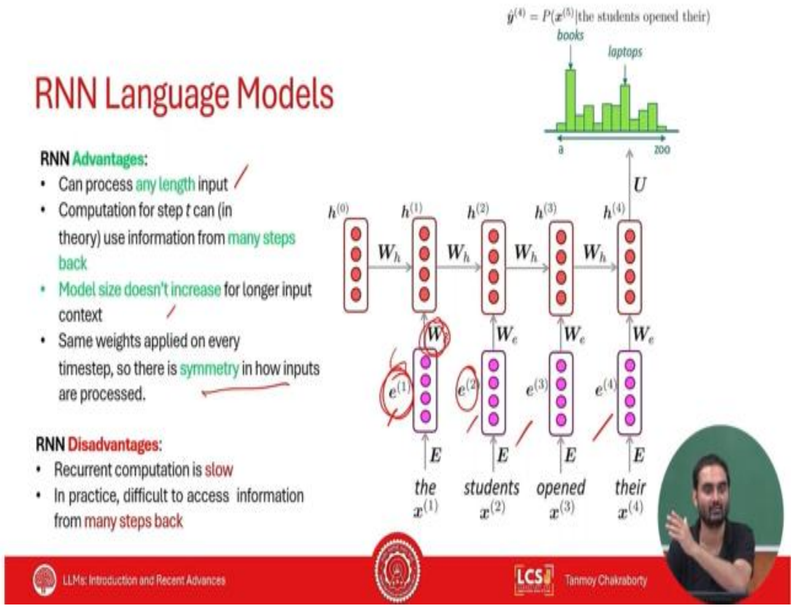
words / one-hot vectors

$x^{(t)} \in \mathbb{R}^{|V|}$

Note: *this input sequence could be much longer now!*

Okay so let's see how you can use RNN to predict the next word, right. So these I already mentioned. Let's say the students open there, okay.

This is the embedding $e^{(1)}, e^{(2)}, e^{(3)}, e^{(4)}$. Right and this is the projection layer we, these are the hidden states, wh is again parameter matrix right and so on and so forth. You can ask what is this e? e is a embedding lookup. Embedding lookup basically says that you have you already have a predefined embedding right and you have an one hot encoding You multiply the one hot encoding with the predefined encoding and you retrieve the corresponding embedding. These are matrix right and this is a vector. Okay so then what you do whatever things you obtain at the last layer which is H4 corresponds to the word there right you pass it through a linear layer you pass it through a linear layer followed by non-linearity same process sigmoid right it will return a distribution and then you sample from the distribution okay.



RNN Language Models

RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input context
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

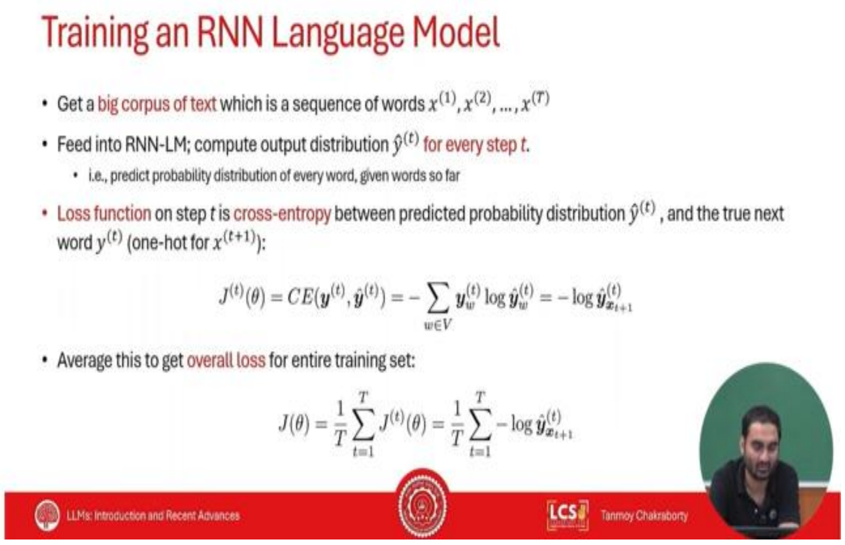
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

So this is RNN, advantage it can ideally process a context size of any right, doesn't depend on the contest length at all I mentioned, right. Ideally, it can also capture things which are far, far from each other. Long distance relation can also be captured. But CNN, remember, CNN can't capture long distance relation. Why? Because CNN only depends on the previous few contexts, previous few words.

But here, ideally, if you look at it carefully, every layer, let's say a t-th layer has the information of all the previous layers from 0 to t-1 with certain changes, right? Ideally, why ideally? I'll tell you later. Model size doesn't increase with the size, with the increase of the context length, right? and this symmetricity right same weights are the same weight if you look at this we this is applied this is multiplied with all the embeddings separately right. Now this guy e1 will affect we similarly to right I mean the effect of E1 to WE and the effect of E2 to WE will be more or less similar. I mean similar in the sense will give fair chance to both E1 and E2. What are disadvantages? The first disadvantage is that this is slow.

Why? Because this is a sequence model. To compute the hidden state at tth time, you need to compute all the previous hidden states. And the second problem is that although I mentioned that it can capture long distance, but as the size of the context increases, we will see the effect of the first word, for example, to the last word will reduce. right. So as we increase the steps in RNN, the effect of the words which are far far from the current word will be less, okay.

Training an RNN Language Model



Training an RNN Language Model

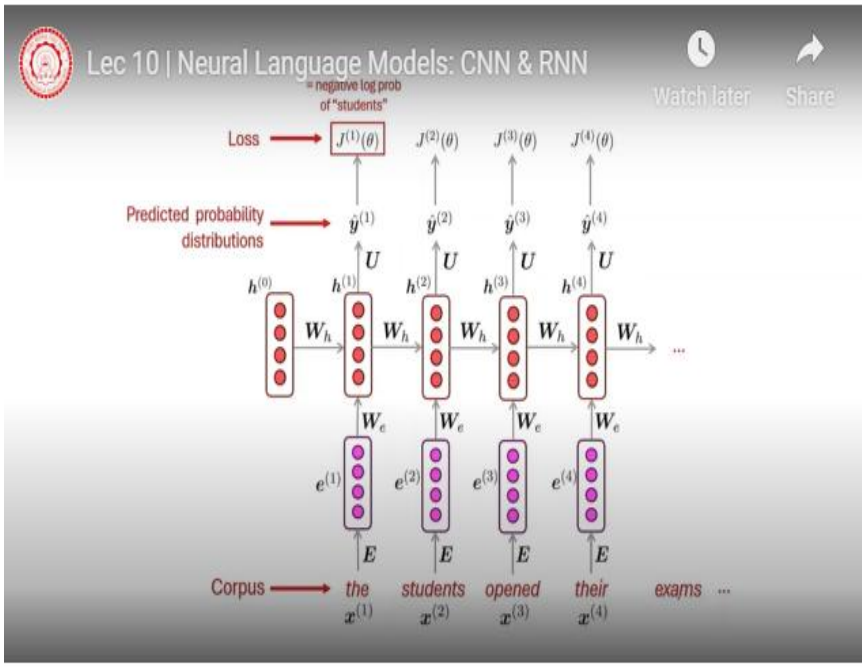
- Get a big corpus of text which is a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
  - i.e., predict probability distribution of every word, given words so far
- Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

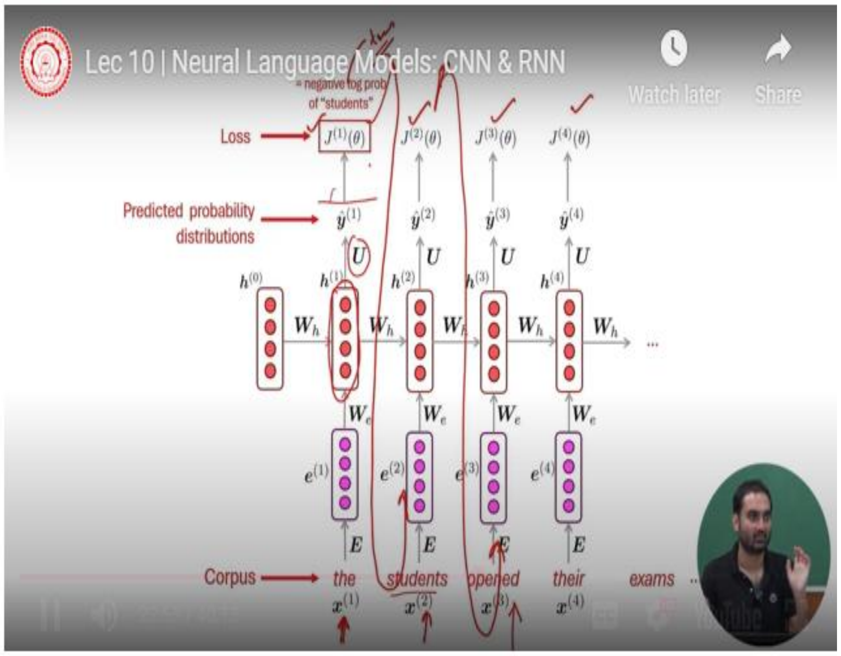
$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

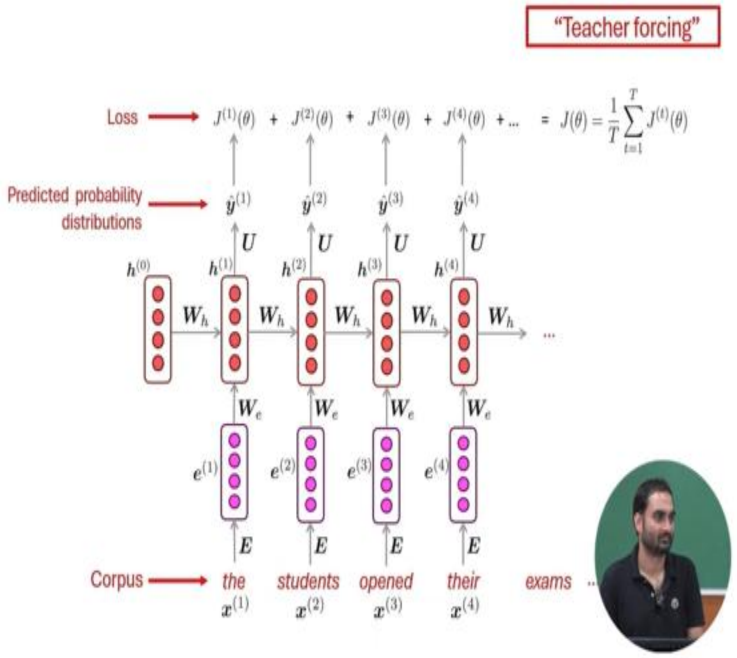
LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty



So now we move to the training part of RNN, okay. So in training an RNN, we need to first look at a loss function, consider a loss function, so now look at this diagram again. The objective is to predict the next word okay. In fact you assume that the objective is to predict the next word at every timestamp right. When you are here your objective would be to predict the next word which is students. When you are here your objective would be to predict the next word which is opened.

when you are here your objective would be to predict the next order which is theirs and so on and so forth right. You also assume that again just for simplicity assume that the input vectors are one hot encoding okay not embeddings obtained from glove or what to get, okay. So here you generate a hidden state the way I mentioned earlier, right? And there is this linear layer u followed by nonlinearity. It will produce a distribution over all the vocabulary words, right? And you choose a sample from it, right? And let's say the sample is teachers. that the sample output is teachers, right? What do you mean by the sample output is teachers? So you basically retrieve the word teacher because its probability was quite high, right? And I also know the one-hot encoding of the word teacher.


Lec 10 | Neural Language Models: CNN & RNN
Watch later
Share
= negative log prob of "students"
Loss
$J^{(1)}(\theta)$
$J^{(2)}(\theta)$
$J^{(3)}(\theta)$
$J^{(4)}(\theta)$
Predicted probability distributions
$\hat{y}^{(1)}$ $\hat{y}^{(2)}$ $\hat{y}^{(3)}$ $\hat{y}^{(4)}$
U
$h^{(0)}$ $h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$
W_h
W_e
$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$
E
Corpus
the students opened their exams ...
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$

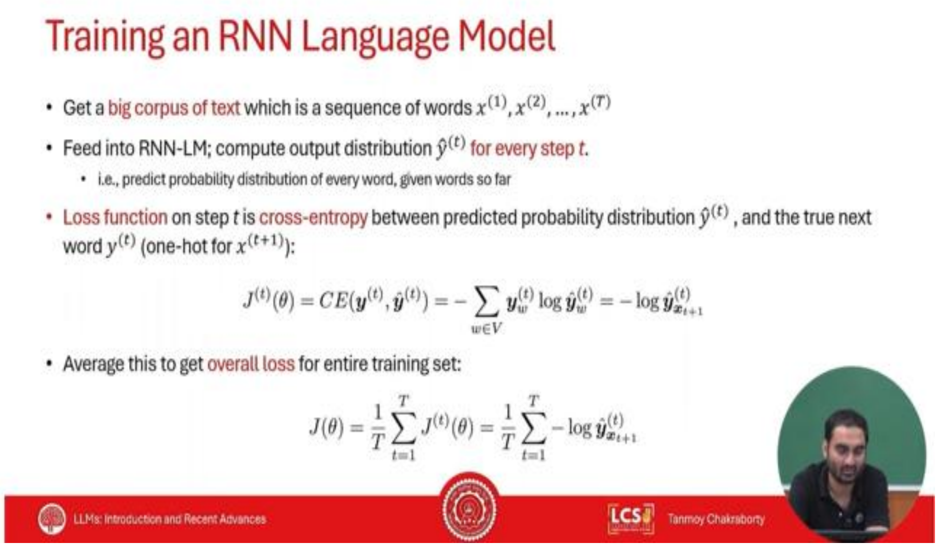

"Teacher forcing"
Loss
$J^{(1)}(\theta) + J^{(2)}(\theta) + J^{(3)}(\theta) + J^{(4)}(\theta) + ... = J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$
Predicted probability distributions
$\hat{y}^{(1)}$ $\hat{y}^{(2)}$ $\hat{y}^{(3)}$ $\hat{y}^{(4)}$
U
$h^{(0)}$ $h^{(1)}$ $h^{(2)}$ $h^{(3)}$ $h^{(4)}$
W_h
W_e
$e^{(1)}$ $e^{(2)}$ $e^{(3)}$ $e^{(4)}$
E
Corpus
the students opened their exams ...
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$

I also know the word not encoding of the word students. So student is my ground truth but the output I obtained here is teachers. So what is the loss? Students embedding and teachers embedding difference. How do we do this? We use cross entropy. So we will measure a cross entropy here. Similarly we measure a cross entropy here, cross entropy here, cross entropy here.

There is a small twist here. When we train, let's say ideally what should happen? Ideally it should be something like that. If the output here is teachers, the teacher output should be fed here, right? And the model should be able to predict the next word as opened. Let's say the model predicted closed. Closed should be given here. So there are two paradigms, right? In one paradigm, whatever output you obtain, you feed that output as an input to the next layer.

The other paradigm is that whatever ground truth you have, input you have, that will be your input to the layer. So if the ground truth is fed to the model, if the output is not fed, the ground truth is fed, it's called teacher forcing, okay? It's called teacher forcing, remember this term. So at every time, at every instance we have this cross entropy. Let's say J1 theta, J2 theta, J3 theta, J4 theta and we take the sum of all these losses which will be my overall loss.

Overall loss is this and we basically take the average of it. There are T number of instances. So this will be 1 by T of total loss.



Training an RNN Language Model

- Get a big corpus of text which is a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
  - i.e., predict probability distribution of every word, given words so far
- Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

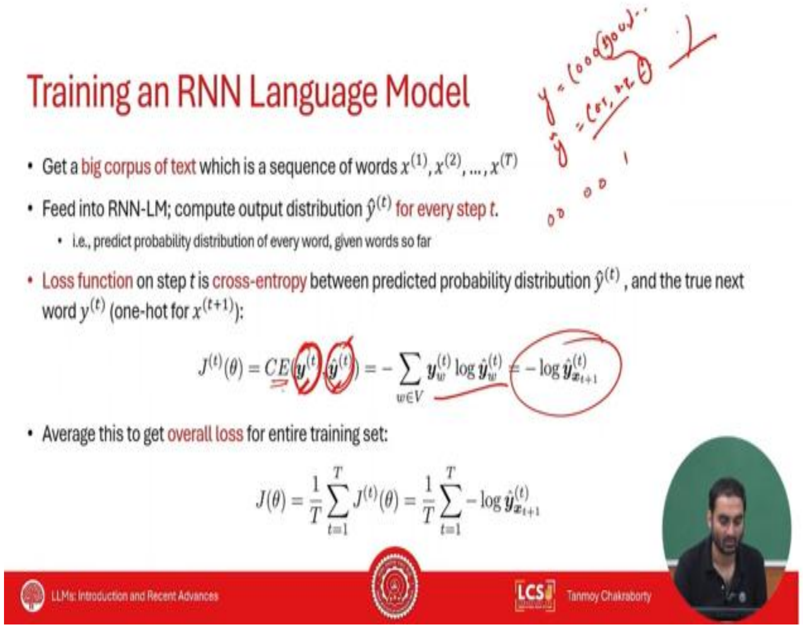
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

LLMs: Introduction and Recent Advances

LCS

Tanmoy Chakraborty



Training an RNN Language Model

- Get a big corpus of text which is a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
  - i.e., predict probability distribution of every word, given words so far
- Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

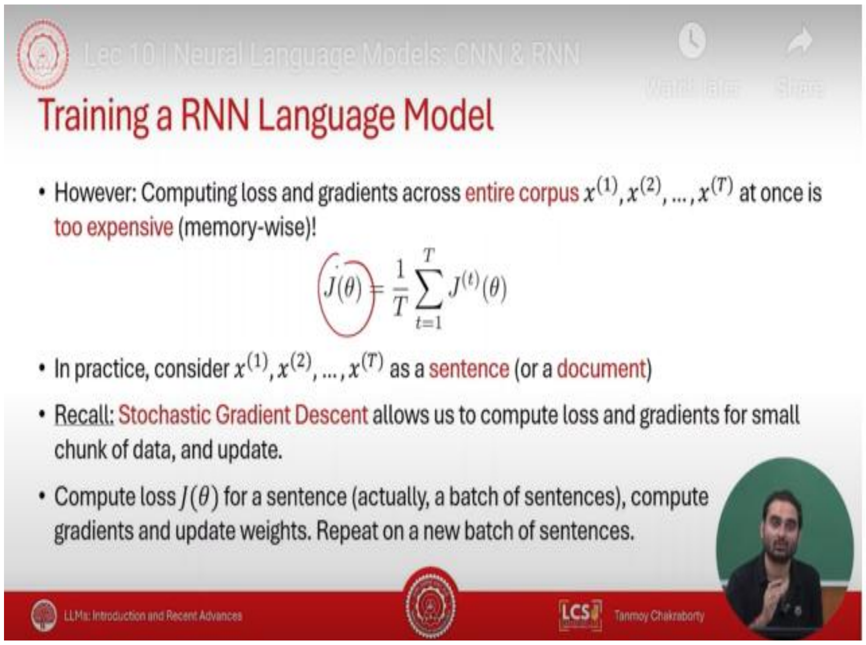
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

Now let's look at the cross entropy a bit carefully. Cross entropy is defined by, cross entropy is already, I mentioned cross entropy right in all the classes. So cross entropy here is basically happening between the predicted embedding and the actual embedding.

Hat is the predicted one and this is the actual one. Okay. So the actual one is one hot encoding. Okay. So it means that all zeros except only one, one, right? So if you do a position wise cross entropy, essentially you will only consider that value of Y hat whose corresponding Y is one. Right so this is y and this y hat y looks like this 0 0 0 1 0 0 0 something like that y hat is 0.5 it is a 0.3 and so on and so forth right so I will only consider this entry and others will be 0 because of this multiplication. So essentially this is minus log of the probability that minus log of the entry that you generated the probability that you generated.



Training a RNN Language Model

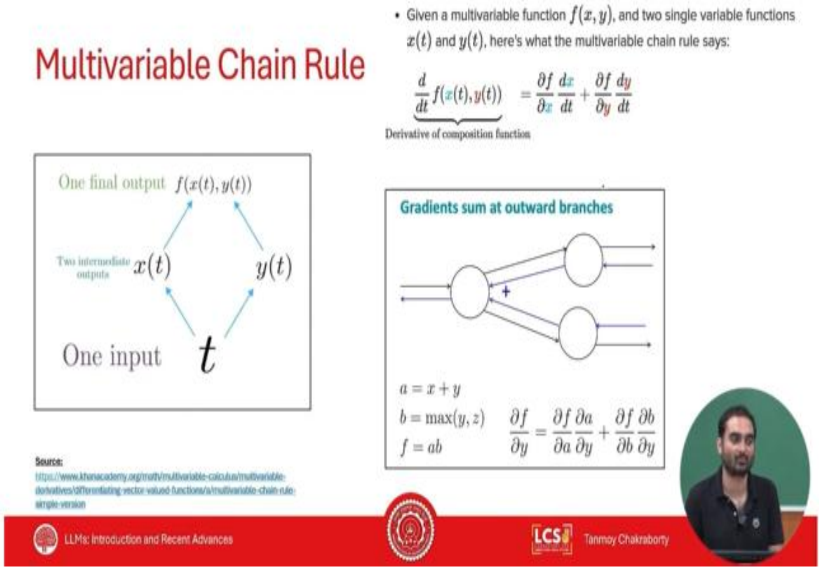
- However: Computing loss and gradients across entire corpus $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$ at once is too expensive (memory-wise)!

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta)$$

- In practice, consider $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$ as a sentence (or a document)
- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.
- Compute loss $J(\theta)$ for a sentence (actually, a batch of sentences), compute gradients and update weights. Repeat on a new batch of sentences.

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

Okay all right so now this is the total loss and we need to and what are the parameters by the way what are my parameters in the RNN? There are three parameters wh we and wu sorry u whatever u or wu, right. So now I need to do backprop.



Multivariable Chain Rule

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

One final output $f(x(t), y(t))$
Two intermediate outputs $x(t)$
$y(t)$
One input t
Gradients sum at outward branches
+

$a = x + y$

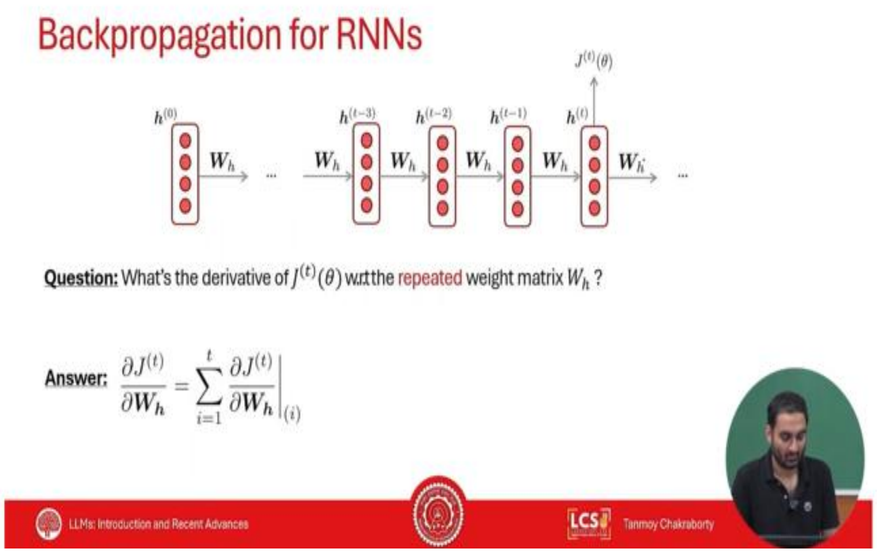
$b = \max(y, z)$

$f = ab$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$

Source: https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty



Backpropagation for RNNs

$$\begin{tikzcd} h^{(0)} \arrow[r, "W_h"] & \cdots \arrow[r, "W_h"] & h^{(t-3)} \arrow[r, "W_h"] & h^{(t-2)} \arrow[r, "W_h"] & h^{(t-1)} \arrow[r, "W_h"] & h^{(t)} \arrow[r, "W_h"] \arrow[u] & \cdots \\ & & & & & J^{(t)}(\theta) \end{tikzcd}$$

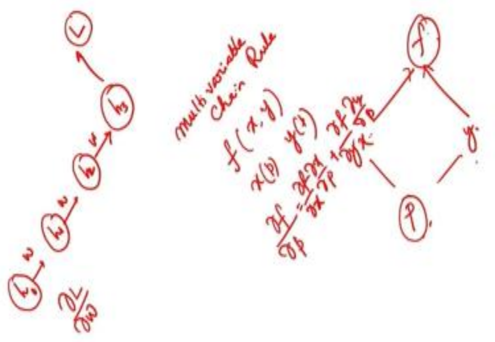
Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t the repeated weight matrix W_h ?

Answer: $\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\Big|_{(i)}$

LLMs: Introduction and Recent Advances

LCS Tanmoy Chakraborty

So in backprop here what I will do? I will use a concept called back propagation through time, it's not a normal back propagation of course the concept is the same, but this is called back propagation through time BPTT okay.



h_0 h_1 h_2 h_3
w
L
$\frac{\partial L}{\partial w}$
Multi variable Chain Rule
$f(x, y)$
$x(p)$ $y(p)$
$\frac{\partial f}{\partial p} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial p}$
f
x
y
p

Let's look at BPTT for the discussion I will assume that there are let's say there are H0, H1, H2, H3 there are four hidden states okay and W is the matrix and this is my loss just a simple diagram forget about WE forget about U because U and WE are very easy to train the main problem the main culprit is W. So this is my loss So when I need to compute this derivative of loss with respect to w, how do I compute? So here I use a concept of multivariable chain rule. What does it say? It says that let's say you have a function f. which is parameterized by x and y and x and y they are also a function of let us say p right. If you want to take a derivative of f with respect to p, how do you do that? So this is my f, it is dependent on x and y and x and y are dependent on p.

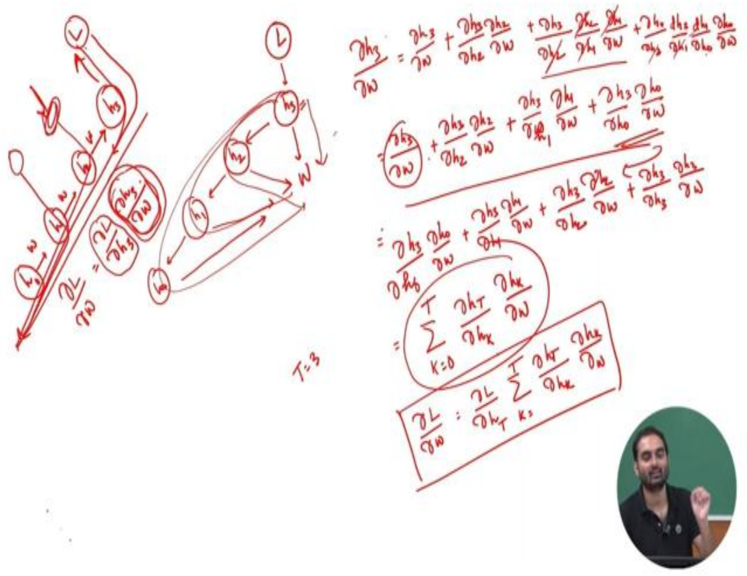
$$f(x, y)$$

$$x(p)y(p)$$

I want to take a derivative of this with respect to this. How do we do this? df dx dx dp plus df dy dy dp. So this will be df dx dx dp plus df dy dy dp. This is the idea.

$$\frac{\partial f}{\partial p} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial p}{\partial y}\frac{\partial y}{\partial p}$$

This is the multivariable chain rule.



Now let us look at this network. and let's assume that let's say you want to measure this derivative dL dW. So to do that I first need to look at the dependence. Dependence is very important.

So this back propagation happens through these directions. right. So, this will be what? This will be dL dH3 right dH3 dW, but H3 is dependent on H2 which is also function of W right. So, this is not that straight forward and this you can easily do, but this is not that easy. So for that let us look at the dependence tree.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial w}$$

So this is the dependence tree. So L, this is L. This is H3. H3 is dependent on W. H3 is dependent on H2 which is dependent on W. H2 is dependent on H1 which is dependent on W. H1 is dependent on H0 which is dependent on W. Okay so to compute del H3 del W we need to look at all possible paths using multivariable channel right. So how do we do that let us only focus on this component okay dH3 dW So dH3 dW is essentially look at this path So this is one path to move to W, this is another path to move to W, this is another path to move to W, this is another path to move to W.

So this will be dH3 dW plus dH3 dH2 dH2 dW. plus dh3, dh2, dh2, dh1, dh1, dw plus dh3, dh2, dh2, dh1, dh1, dh0, dh0, dw.

$$\frac{\partial h_3}{\partial w} = \frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial h_0}\frac{\partial h_0}{\partial w}$$

Let us do some small changes here and there dh dW plus dh3 dh2 dh2 dW plus what is this one? Can I simplify this? Reason this dh3 dh1 dh1 dW. plus now look at this one do dh3 h2 h2 h1 h1 h0 right h2 h2 will cancel out h1 h2 will cancel out this will be dh3 dh0 dh0 dw okay.

$$= \frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w} + \frac{\partial h_3}{\partial h_0}\frac{\partial h_0}{\partial w}$$

Now let us see what is actually happening? can we think about genetic equation out of this let us look at this let us write the entire equation in a reverse this one will come first.

H3 dH1 dH1 dW plus dH3 dH2 dH2 dW plus dH3 dH3 dH3 dW, this term.

$$= \frac{\partial h_3}{\partial h_0}\frac{\partial h_0}{\partial w} + \frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial w}$$

Right so here let us say t equals to 3 the third stage right can I write it as sum of k equals to 0 to t right dHt dHk dHk dW. right.

$$T = 3;$$

$$= \sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k}\frac{\partial h_k}{\partial w}$$

So, this is you know the actual derivation for this one. So, ultimately what is del L del W? This is del L this is del HT sum of k equals to let us make it generic ok.

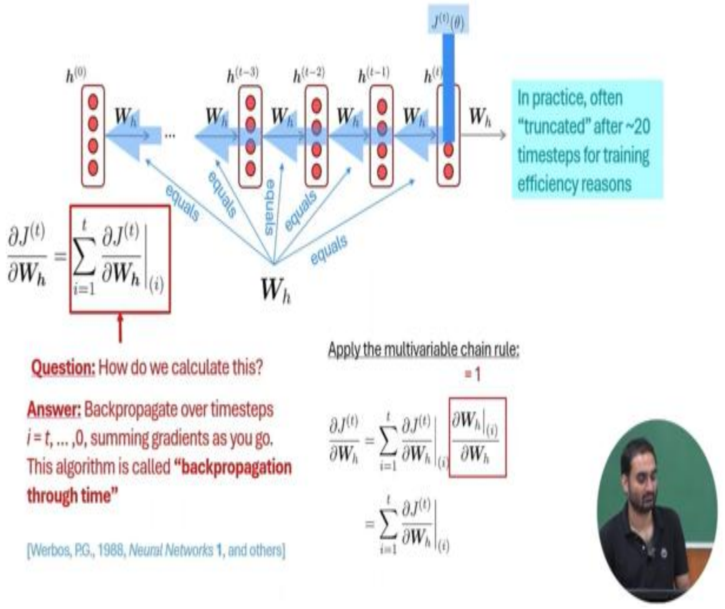
So, k equals to 0 to t del ht, del hk, del hk, del w.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_T}\sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k}\frac{\partial h_k}{\partial w}$$

This is the final formula. This is called back propagation and the other you know when you take the derivative with respect to u with respect to w this would be straight forward because I mean you can also do it in a similar way for example. Okay now using this formula you update the value of w okay. Now what is actually happening here think about it now in a very high level. last layer, you compute the loss, right? And that loss gets back-propagated and the W vectors, the W matrix will be updated.

Similarly, you have another loss in the second position, right? That will also affect W. When you back-propagate, let's say here, you back-propagate through this, W will be updated. You also have a loss here. Isn't it? We have a loss here, this will also be back-provocated. We have a loss here, this will also be back-provocated.

So every position will affect W and W will be updated multiple times. Right?



In practice, often "truncated" after ~20 timesteps for training efficiency reasons

$$\frac{\partial J^{(t)}}{\partial W_h} = \boxed{\sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps $i = t, \dots, 0$, summing gradients as you go. This algorithm is called **"backpropagation through time"**

[Werbos, P.G., 1988, *Neural Networks* **1**, and others]

Apply the multivariable chain rule:

$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)} \overset{= 1}{\boxed{\frac{\partial W_h|_{(i)}}{\partial W_h}}}$$

$$= \sum_{i=1}^{t} \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}$$

Okay, now think about it. What's the problem here? So if you look at it very carefully, this one by the way here I you know I is the derivative is not so easy by the way because these are not normal derivatives remember h is a vector and ht is a vector hk is also a vector right w is a matrix. okay. You should know how to take a derivative with respect to a matrix, how to take a derivative of a vector with respect to a matrix, derivative of a vector with respect to a vector. You should thank Hugging face and all these libraries right that you do not need to do all these things you know manually okay.

So now let me write this one again. dL dW is dL dHt sum of k equals to 0 to t dHt dHk dHk dW. Look at this part.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_T} \sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k} \frac{\partial h_k}{\partial w}$$

So, what is this? Think about it so these are hidden states okay H1, H2, H3 right let us say H4 and so on and so forth and you have a loss here L okay so by the way how do you compute this loss dL dHt? Let us say dL dH1 how do you compute dL dH1 dL dH1 is dL dH4 times dL dH4 dH3 dH3 dH2 dH2 dH1 okay. So this with respect to this when you take the derivative you are multiplying multiple such derivatives. Now each of this derivatives which looks like this dK d dHK dHK minus1 right, each of these derivatives is essentially a culprit. So if now by the way you are taking a derivative of a vector with respect to another vector it will produce a matrix right. So if the value of the matrix I will tell you what is the meaning of a value of matrix. If a value of a matrix is low and you keep on multiplying multiple such low valued matrix, multiple such low values, small values and you have multiple steps let's say from H1 so you start with H1 and then let's say H80. If you keep on multiplying this small values you will see that this will be even lower and lower.

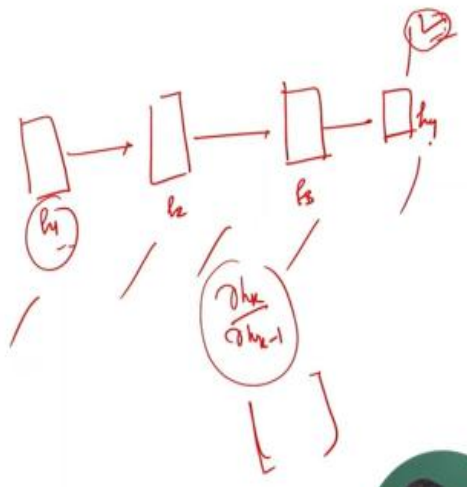
okay. What is the meaning of the small value of a matrix? You take the largest eigenvector, eigenvalue corresponding to the largest eigenvector and if you see this is less than 1 then it is a small matrix, okay. You can prove this also. So intuitively if it is a small matrix then we see the effect of the first hidden state which is H1 on the loss L will be much, much lesser than let's say the effect of the last hidden state on the loss. And this is called the vanishing gradient problem.

With the increasing depth of the RNN, you keep on forgetting the information that you had earlier. And if the matrix is large, eigenvalue is greater than what? Exploding gradient problem. Exploding gradient can be addressed using gradient clipping. You can at every stage you can say that look I don't want the value to be greater than some threshold. You can clip the gradient. And vanishing gradient how do you address? By the way the BPTT method I mentioned back propagation through time there are multiple variations of it.

There is something called truncated back propagation. Truncated BPTT. Truncated BPTT you don't allow this to move till the end. okay. You stop somewhere and you decide when to stop.

I mean how much you allow the gradient to propagate, to move. You do not allow it till the end, okay. So this is about backprop. In the next class we will discuss how we can address the vanishing gradient problem using advanced Rn methods like LSTM and GL. Thank you.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h_T} \sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k} \frac{\partial h_k}{\partial W}$$



h_1
h_2
h_3
h_4
L
$\frac{\partial h_k}{\partial h_{k-1}}$