Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
LSTMs and GRUs

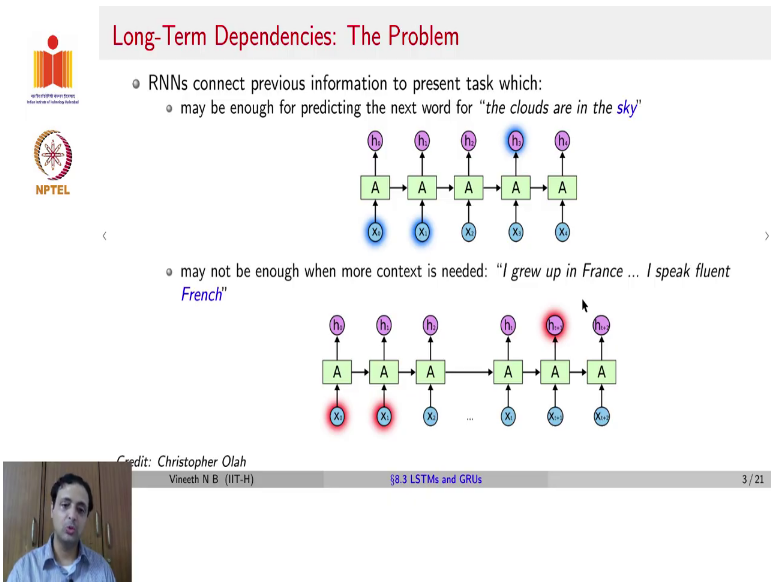
(Refer Slide Time: 0:19)

We will now talk about how to solve the vanishing gradient problem using changes in the architecture of an RNN, in particular we will talk about LSTMs and GRUs.

(Refer Slide Time: 00:31)

One question that we left behind was, how do you tackle the vanishing gradient problem in RNNs without changing the architecture? You could do a few things, you could use ReLu instead of tanh and sigmoid. Remember, ReLu does not vanish the gradient, it keeps the gradient at least the positive, for the positive activations it keeps the gradient as it is, you could regularize in some way, you could initialize the weights in some way that helps carry forward the gradients backward and you could also consider using only short time sequences so that the gradient does not vanish within that short time frame.

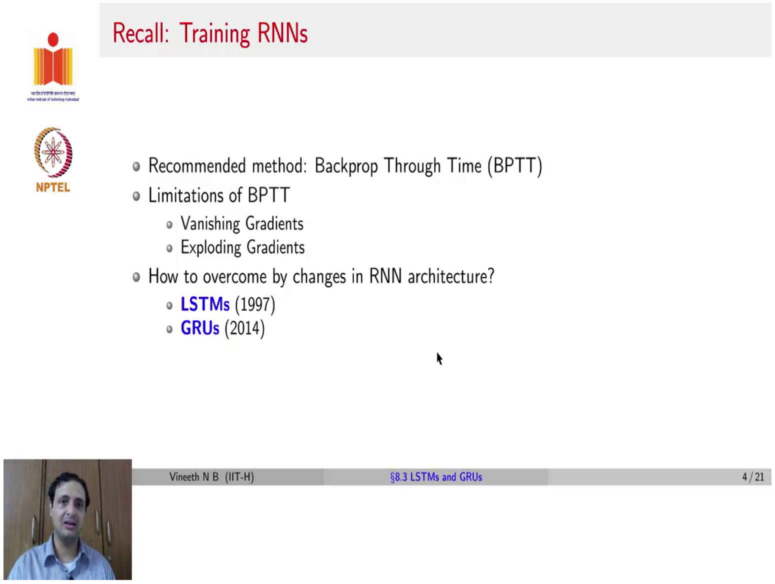
(Refer Slide Time: 01:18)



Let us revisit this problem of long-term dependencies. So, if you had a situation where you were dealing with the sequence learning problem, which say wants to predict a sentence the clouds are in the sky or it wants to classify say the sentiment or the category of the sentence, an RNN may be good enough because the word sky may only depend on the word cloud which is say just two words before if you remove all the basic words in the phrase.

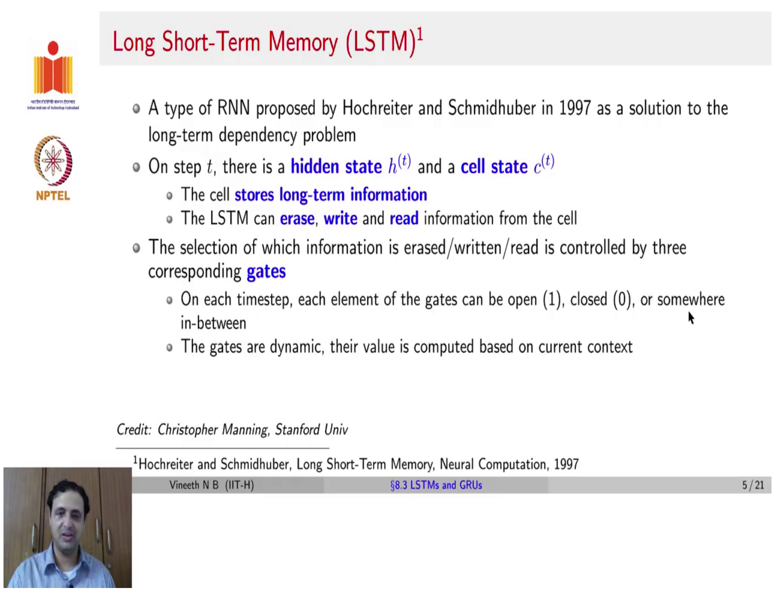
However, if you have a more detailed sentence, a longer sentence such as I grew up in France, I speak fluent French. Now, the word French at a particular time step may depend on some other word that was used several words ago. These long term dependencies may not really get captured by an RNN, because the presence of the word France may not really have an impact on the word French because the gradients may vanish between these two time steps. So, we are going to talk about how we address this in this lecture.

(Refer Slide Time: 02: 40)



Just to recall RNNs are trained using back propagation through time. The limitations of back propagation through time that we discussed were vanishing gradients and exploding gradients. Exploding gradients we said we could handle using gradient clipping. How do you now address vanishing gradients using changes in RNN architecture are through LSTMs and GRUs as examples and we will discuss both of them in detail in this lecture.

(Refer Slide Time: 03:15)



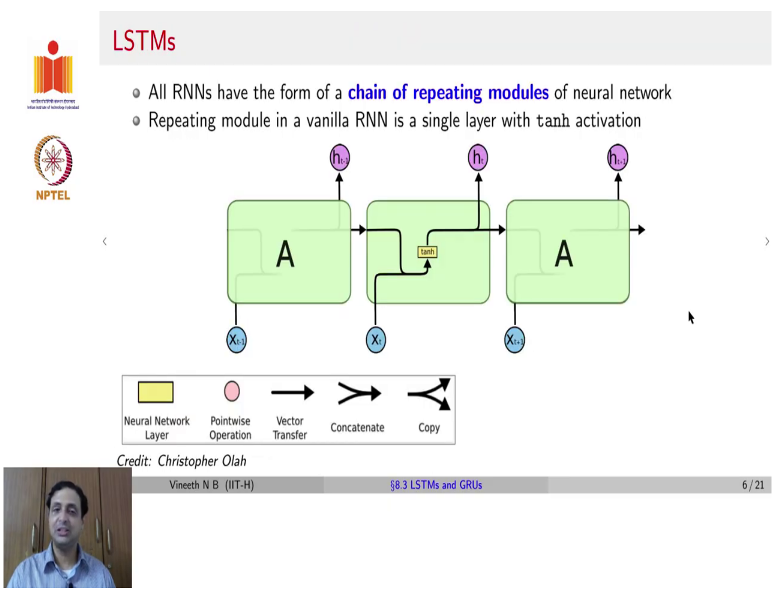
LSTM stands for Long Short-Term Memory. It was introduced way back in 1997 by Hochreiter & Schmidhuber. Primarily intended to address this long-term dependency problem although it was not in the same form that we will discuss now, but we will also discuss its evolution as we go forward.

So, here is the design of an LSTM. At each step t, in an RNN we had a hidden state which the RNN block would output. Now, in an LSTM, we would have a hidden state which an LSTM block would output and also an internal cell state which maintains the information across a temporal context. The cell stores long-term information and the LSTM block can erase, write and read information from that cell state based on whatever context defines.

The selection of what information you can forget or read or write is controlled by three gating mechanisms, one for erasing, one for writing and one for reading. And on each time step these gates could assume values open which would be 1, which would allow all the information to pass through, closed which would be 0 which would not allow any information to pass through or somewhere in between.

You can also have an information lying between 0 and 1. These gate values are dynamic and are learnt and computed based on the input at a particular time step and the hidden state that comes from the previous time step. Let us see this, each of these components in more detail.

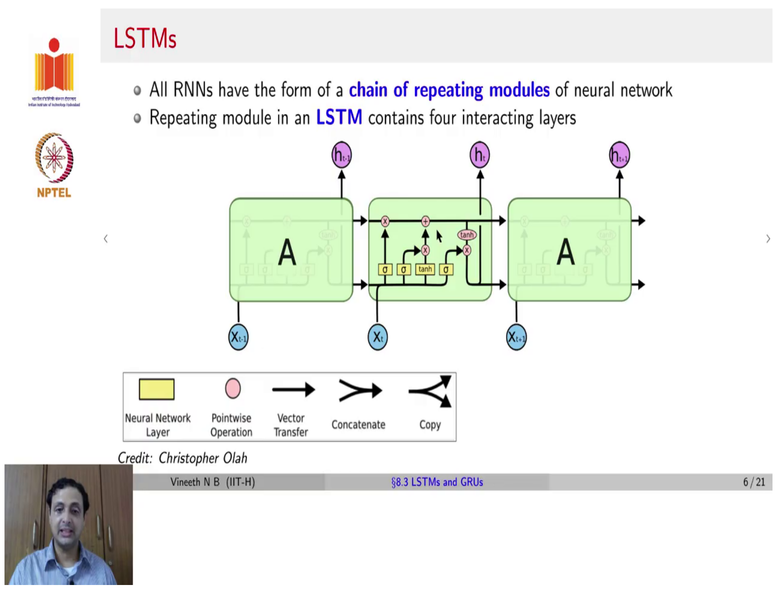
(Refer Slide Time: 05:20)



A RNN has a general form of a chain of repeating modules. So, if you took a vanilla RNN the repeating module is perhaps a single layer with the tanh activation function. As we said earlier, it can also be not just a single layer but two, three layers, but that would be the repeating block that is applied to each, to the input at each time step.

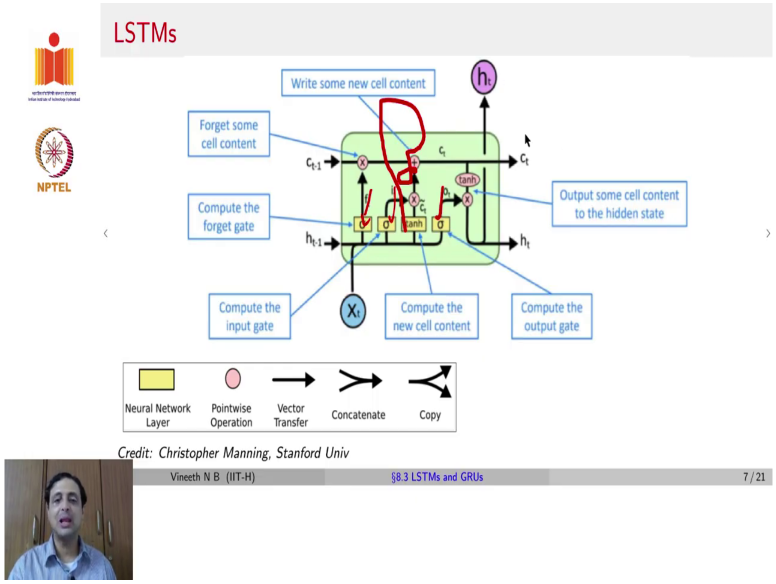
So, this diagram here you have an input X_{t-1} at t-1, X_t at time t and X_{t+1} at time t+1 and what you see here is a single neural network layer which has a tanh activation function and the input at a particular time step X_t as well as h_{t-1} which is the output of the previous RNN block, is provided as input, a tanh is applied that gives us h_t and that h_t is given as output as well as given to the next RNN block at the next time step. That is the vanilla RNN that we have seen so far drawn in a different way. Let us now try to draw a parallel with LSTMs.

(Refer Slide Time: 06:44)



LSTMs have a similar structure. Once again it is a chain of repeating modules where you apply the same block at every time step. But now the structure of each block is not just one layer or two layers. It is different. It contains four different layers and they are not sequential layers unlike the networks we have seen so far. They also interact with each other. We will see each of these layers in detail.

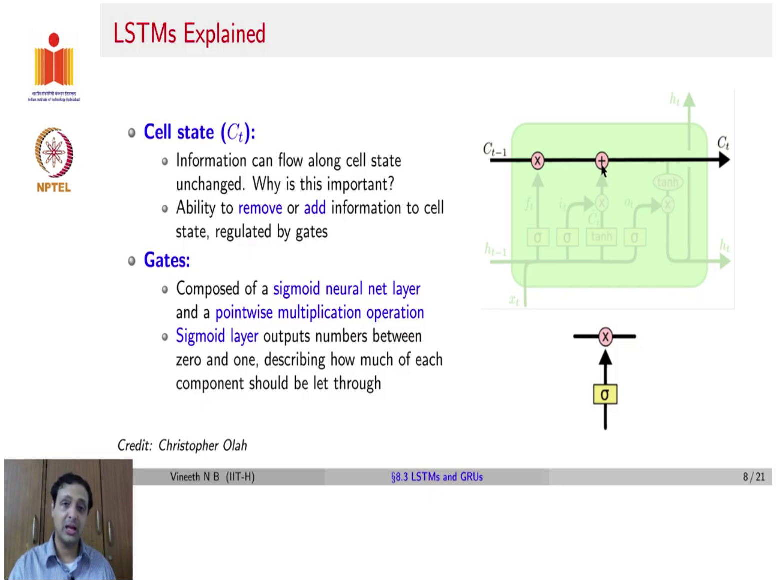
(Refer Slide Time: 07:17)

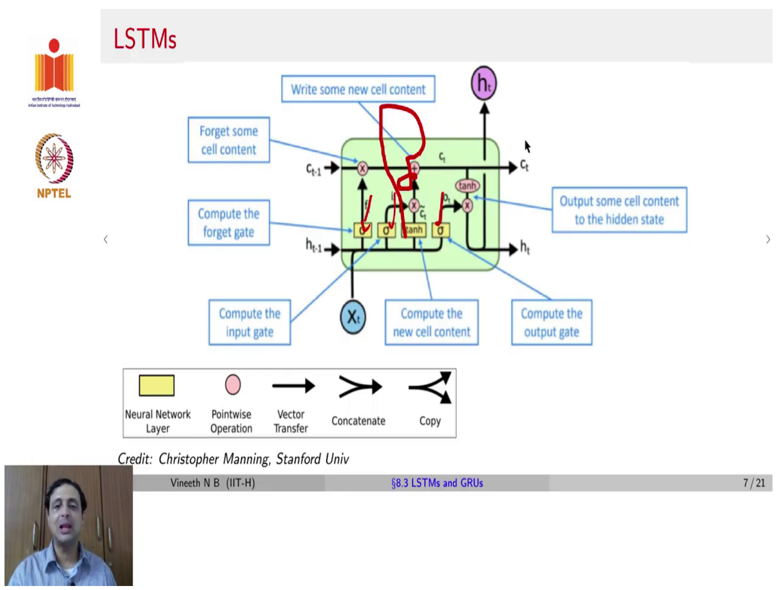


So, in this case, you can see that you have four different layers denoted by these four blocks, four yellow blocks. One of them is known as a forget gate which decides to forget some cell content coming from the previous state. Another of them is known as an input gate and that is why you see a sigmoid activation function here given by σ. So, these ones are sigmoid activation functions. And the reason we use sigmoid here is because we want the output to lie between 0 and 1.

So, the input gate along with a tanh activation function decides how much of the input should be written and also adds that new cell content at the end. So, that is what happens here. And finally there is an output gate which decides how much of the cell state should be exposed as the hidden state. So, cell state is denoted as C_t and the hidden state is denoted as h_t and a part of the cell state is revealed as hidden state to the next time step as well as an output of the cell for further processing. Let us see each of these in more detail.

(Refer Slide Time: 08:46)



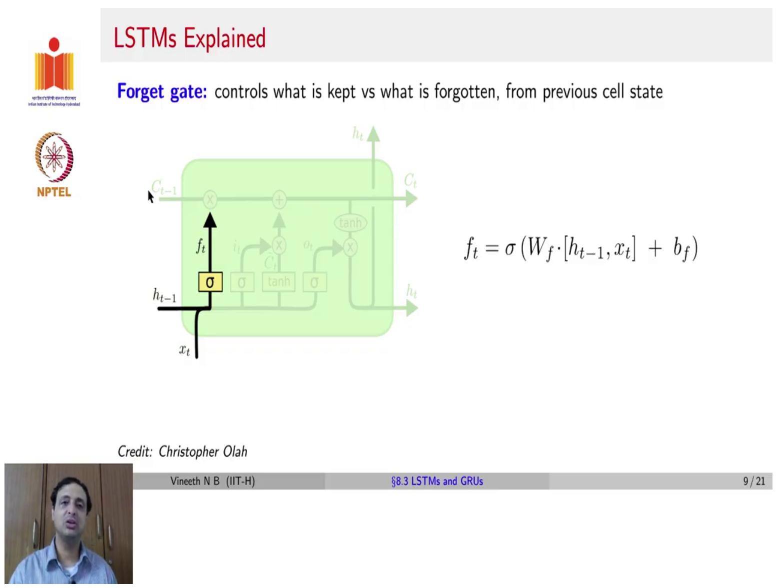


So, the cell state contains information which you can look at as a memory across a temporal context and the information can flow across the cell state unchanged. If you see this diagram here, this is just the diagram from the previous slide with a few components grayed out for the sake of explanation. So, you can see here that the information in a cell state C_t passes as it is to the previous cell state C_{t-1}. Why is this important? Try to connect this to the vanishing gradient and we will come back and talk about this point later in this lecture.

The only thing that we have is a multiplication operation here and an addition operation here, the multiplication operation decides whether you want to forget something from a previous cell state C_{t-1} that contains a sigmoid neural network layer, which has a dimension as the size of the cell state and has a value between 0 and 1 for each dimension of that cell state.

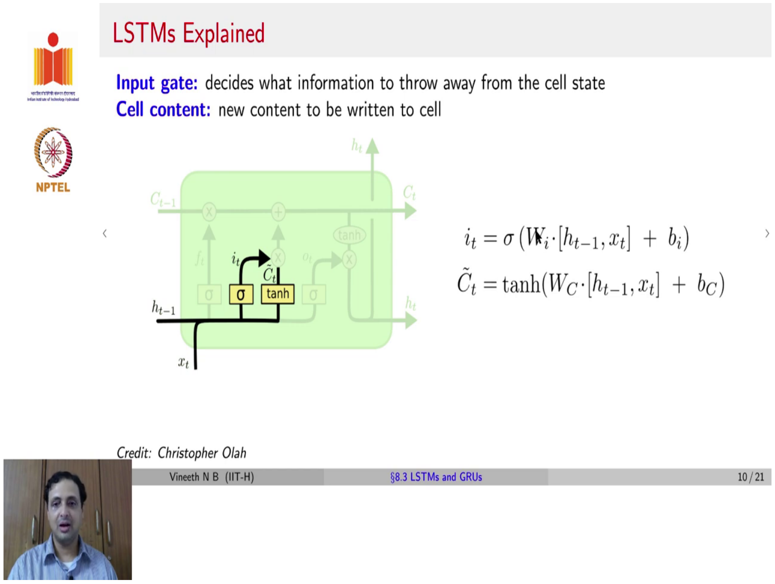
So, for each dimension of the cell state you can decide how much of that information you want to retain using this multiplication operation here. This multiplication operation is a point wise or element wise multiplication operation. Then you also have an addition operation here which decides how much of that information gets added to a, how much of an information coming in gets added to this new cell state.

(Refer Slide Time: 10:36)



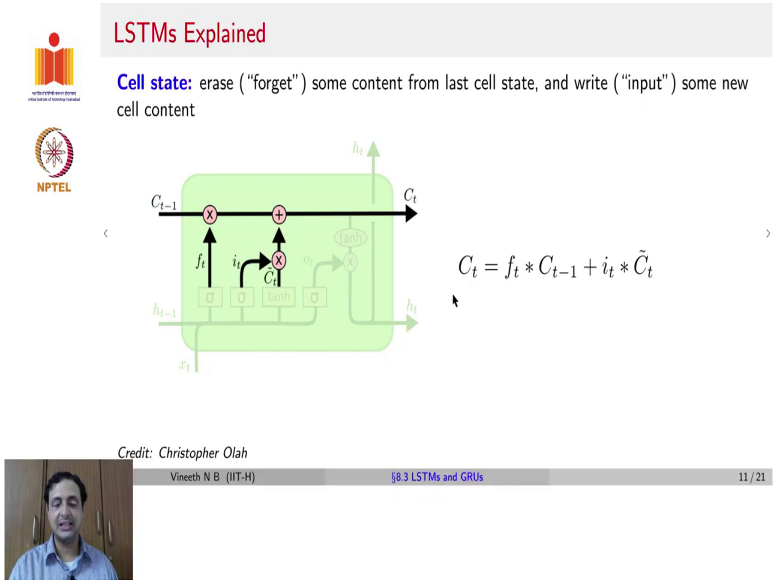
Now, let us talk about the forget gate to start with. The forget gate, which is one of the layers inside the LSTM block takes input x_t, takes input h_{t-1} very similar to the RNN block that we saw so far. Then it has a set of learnable weights W_f, those W_fs are learnt while training you have a bias b_f corresponding to the weights. And then you apply a sigmoid activation function to ensure your outputs lie between 0 and 1 that is the output of the forget gate. And what does the forget gate do? It does an elementwise multiplication with the previous cell state C_{t-1} to decide how much of that information should be erased and how much of that information should be kept.

(Refer Slide Time: 11:32)



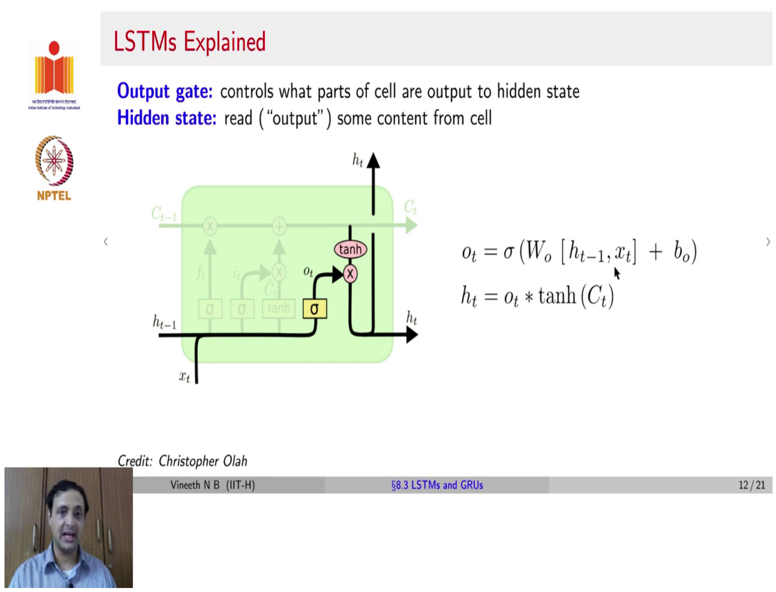
Similarly the input gate decides what information to remove from the cell state. So, the input gate also receives a copy of x_t and h_{t-1} as input similar to forget gate has its own weights W_i, a bias b_i and operates a sigmoid activation function to ensure the output lies between 0 and 1. And in this particular part of the architecture, you also have another component which takes the same input h_{t-1}, x_t. Remember, those are the only two inputs that you get to an RNN block. It is the same inputs here, which has its own weights W_C, a bias b_C applies a tanh activation function so that you can have both negative and positive values and you get an output $\overline{C}_t$

(Refer Slide Time: 12:34)



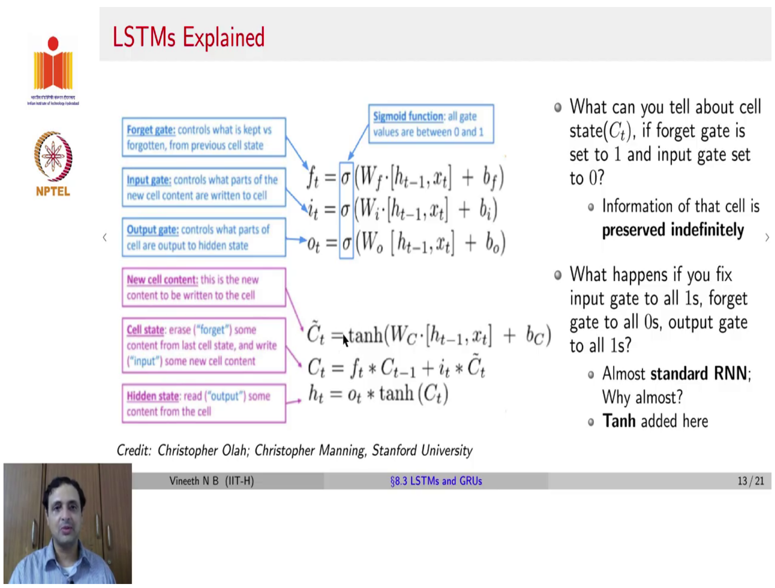
Let us see how these are combined now. So, the final cell state Ct is given by $f_t * C_{t-1}$ So, this tells you which dimensions of the previous cell state C_{t-1} should be forgotten and to what extent and $i_t * \overline{C_t}$ to decide what should be written on to the current cell state. So, you have a cell state which is like a memory, in the current time step you want to decide what aspect of the memory do you want to remove and what new content do you want to add to the memory.

(Refer Slide Time: 13:16)



Finally, you have the output gate which controls what parts of the cell state are provided as the hidden state which goes to the next time step and the output. So, o_t is another gate similar to the forget gate and the input gate, once again receives h_{t-1} and x_t as input, has its own weights W_0 and b_0, sigmoid activation function because it is also a gating mechanism and this is now combined with C_t which is the current cell state element wise to decide what should be the h_t that is provided to the next cell state, next state as well as to the output.

(Refer Slide Time: 14:09)



Now, to summarize these in equations. So, f_t is a forget gate which controls what is kept versus what is forgotten from the previous cell state. The input gate controls what parts of the new cell content are written to the cell. The output gate controls what part of the cell are output to the hidden state. All of them use a sigmoid activation function so that the output is similar to a gating mechanism and lies, the values lie between 0 and 1.

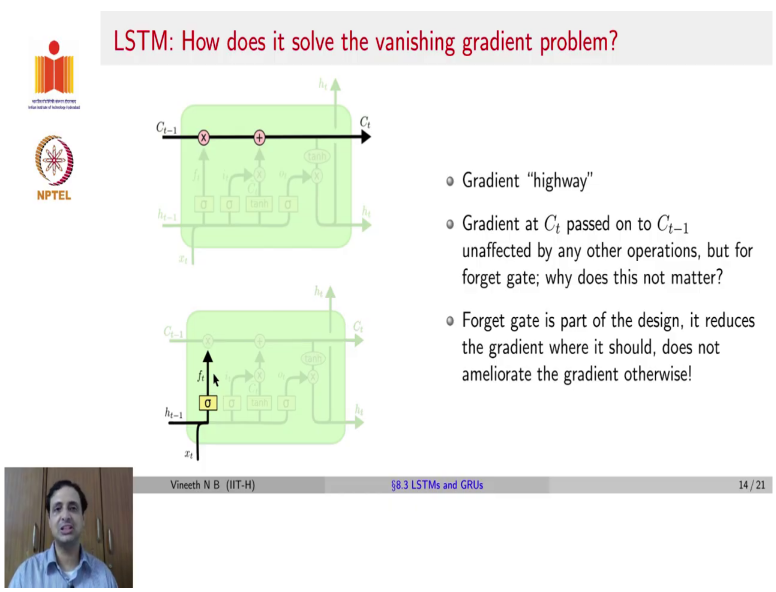
And $\overline{C}_t$ is the new cell content that you want to write to the cell. So, that is just a simple processing of the inputs and C_t finally is obtained by forgetting some content from the previous state and writing some new cell content f_t controls C_{t-1} and the input gate i_t controls the $\overline{C}_t$ which you ideally want to write onto the cell state.

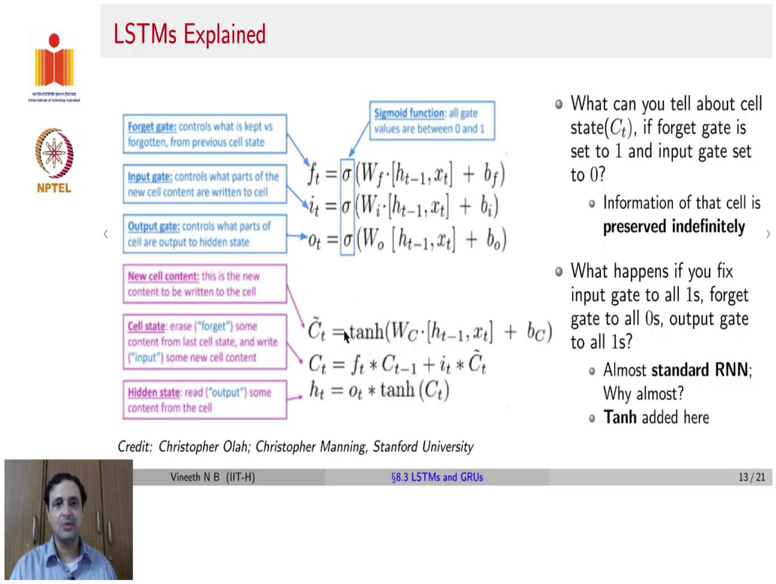
And finally the hidden state is and decides to read some content from the cell as output using the output gate. It again uses a tanh to maintain negative and positive values. Let us ask a couple of questions here to understand how this entire setup works. What can you tell about the cell state C_t, if the forget gate is set to 1 and the input gate is set to 0.

Let us try to analyse this, the forget gate is set to all 1s and the input gates is set to 0s. What would happen from this equation here for C_t you can notice that the information of the cell coming in from the previous cell state will continue to be preserved because there would be no input that would be added due to the input gates being 0 and the $\overline{C}_t$ would not have effect on the cell state.

Let us ask another question, what would happen if you fix input gate to all 1s, forget gate to all 0s and output gate to all 1s? If you thought carefully this will almost be the standard RNN think about it to ensure you can understand this. Why do we say almost standard RNN? The only difference now is there would be a tanh that gets added here, which was not there in the vanilla RNN.

(Refer Slide Time: 17:02)





Now, let us come back and ask this question. It seems like a complex architecture. One point to add to the discussion so far is that these four layers in an LSTM are not necessarily sequential the way we have seen networks so far. We of course saw skip connections where any layer could be connected to any other layer, similarly even in an LSTM there are interactions between the layers to achieve a specific objective.

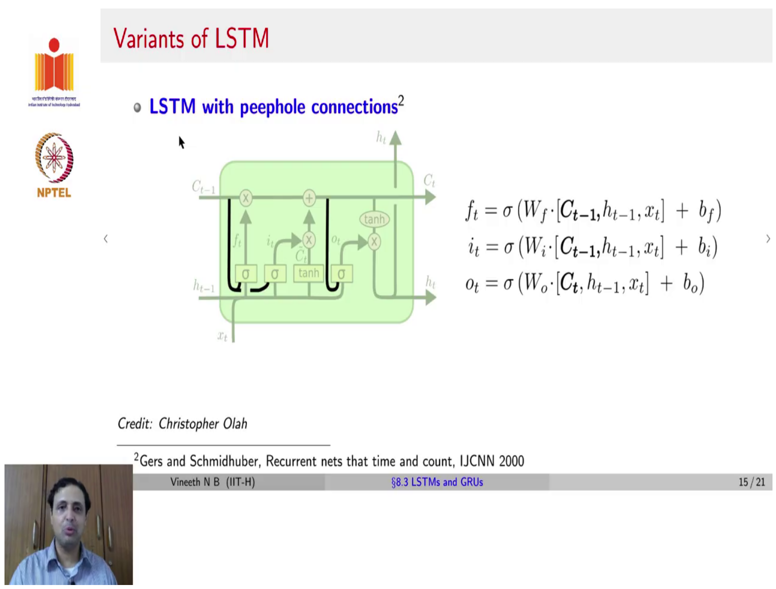
Now, let us try to ask. How does the LSTM really solve the vanishing gradient problem? The key to that lies in this highway here between C_t and C_{t-1}. We are going to call that the gradient highway. So, whatever gradient of the error that you receive at C_t will be passed on as is to C_{t-1}.

Earlier we had to worry about that being mitigated by the activation function across a layer so on and so forth. Here if you notice between C_{t-1} and C_t there is no layer parse, the only thing that exists between them is the output of the forget gate, which is just a vector, it is not a layer, there are no weights, it is just a vector. Why is this not a problem? This is not a problem because the job of the forget gate is to decide how much C_{t-1} should contribute to C_t.

So, if the gradient is reduced to a previous state based on how much it contributed that is a fair gradient. We will not be worried about it. The gradient does not get mitigated because of any other operations in the LSTM. It depends only on the forget gate and the forget gate's job is to control how much of the previous cell state goes to the next cell state. So, the gradient would only get mitigated by that much amount for each of the dimensions in the previous cell state. This allows LSTM to solve the vanishing gradient problem.

So, once again, if you go back to the equations here this equation on C_t tells us that C_t, it depends on C_{t-1} only with respect to f_t while the second term here the $i_t * \overline{C_t}$ could still be affected by the vanishing gradient problem. We do not worry about it because there is another component which will allow the gradient to pass through as is through the LSTM network to earlier time steps. This allows addressing the vanishing gradient problem in LSTMs.

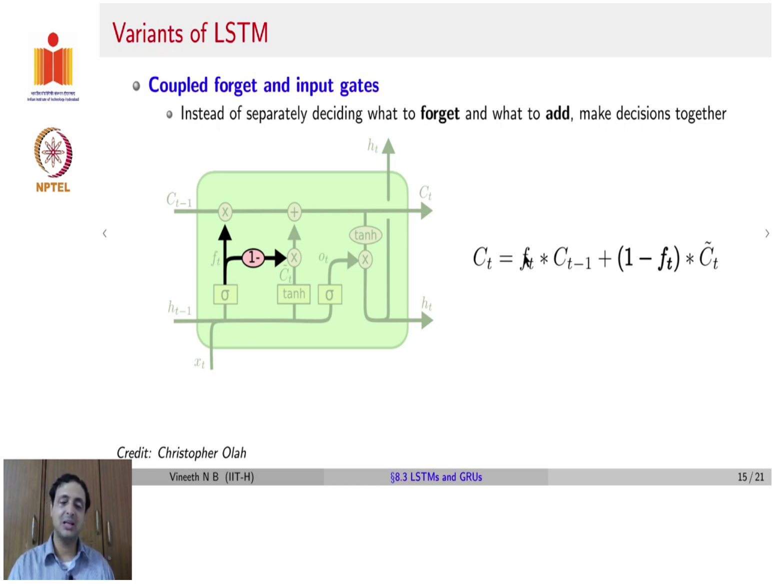
(Refer Slide Time: 19:56)



Over the years since LSTMs came way back in 1997 there have been a few variants of LSTMs that have been developed, a couple of popular variants are, one of them is known as an LSTM with peephole connections. The reason it is called an LSTM with peephole connections is in the computation of the forget gate, input gate and output gate you notice that in addition to h_{t-1} and x_t you also provide C_{t-1} as inputs and for the output gate you also give C_t as input.

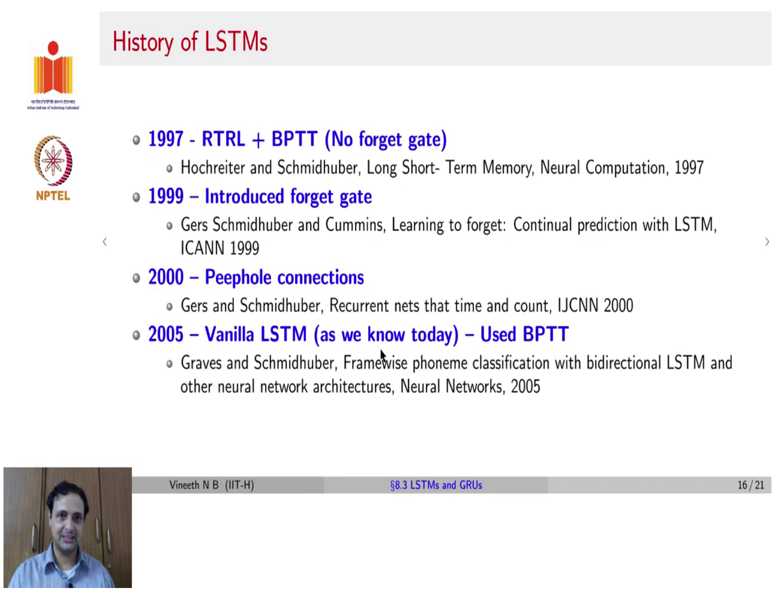
So, you are allowing your gates to peep into the cell state and then decide which dimension you should cut down or which dimension you should let through. That is why these are known as LSTMs with peephole connections.

(Refer Slide Time: 20:57)



Similarly, there is also been another variant known as an LSTM with coupled forget and input gates where instead of having a separate forget gate and an input gate the cells state is computed as $f_t * C_{t-1}$ and $(1 - f_t) * \overline{C_t}$, we had an i_t in the vanilla version of the LSTM, but in this version the f_t doubles up to serve both these purposes. So, f_t tells you how much to forget and $1 - f_t$ tells you how much to let it. This is a variant that has been proposed.

(Refer Slide Time: 21:39)



So, the history of LSTM to summarize is in 1997 Hochreiter and Schmidhuber first proposed LSTMs and the learning at that time used was known as real-time recurrent learning with backpropagation. There was no forget gate in this version of the LSTM in the late 90's. Then in 1999 Schmidhuber again their group introduced the forget gate into the LSTMs, in 2000 the variant with peephole connections was introduced. And in 2005 Alex Graves who is probably responsible for the version of RNNs and LSTMs that we see today introduced the vanilla LSTM as we know today with all the components.

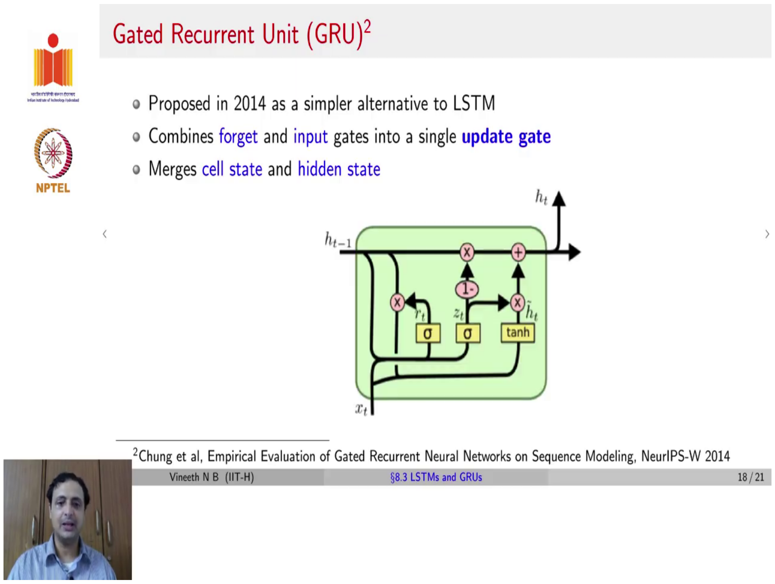
(Refer Slide Time: 22:28)

Over the last few years especially between 2013 to 15 LSTMs started achieving state of the art results on various applications such as handwriting recognition, speech recognition, machine translation, parsing, image captioning so on and so forth that they started becoming the default choice for doing sequence learning.

However, while we will not have the opportunity to discuss this now, in 2020, as we speak, One of the hottest trends to handle sequence learning problems are known as Transformers. Transformers use the idea of what is known as self attention. In fact in a recent competition a summary report of all the methods that participated in that competition WMT stands for machine translation. It is a machine translation which is a sequence learning problem.

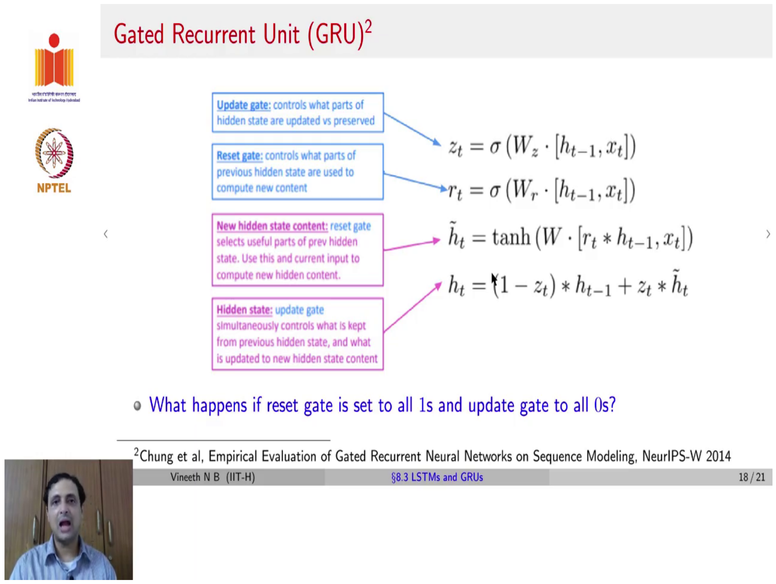
When we say machine translation, We mean an application such as Google translate to go from English to German or Hindi to English, so on and so forth. So, with all the participants that provided entries to this competition only 7 of them were RNN based, while 105 of them were transformer based. That should give you an idea of what is the latest trend at this time.

(Refer Slide Time: 23:51)



In 2014 there was another variant similar to LSTM known as the gated recurrent unit GRU, which was also proposed, which is also used popularly as an alternative for LSTM today. It was proposed by Chung et al. So, the main idea here is very similar to the coupled forget and input gates. GRUs combined forget and input gates into a single update gate. It also merges the cell state and hidden state and this is the overall architecture. Let us see this in some more detail.

(Refer Slide Time: 24:31)



So, in this GRU you have only two gates instead of an input gate, a forget gate and an output gate. Now, you have only two gates. These gates are called an update gate and a reset gate. So, these look exactly the same as any other gates that we saw with an LSTM.

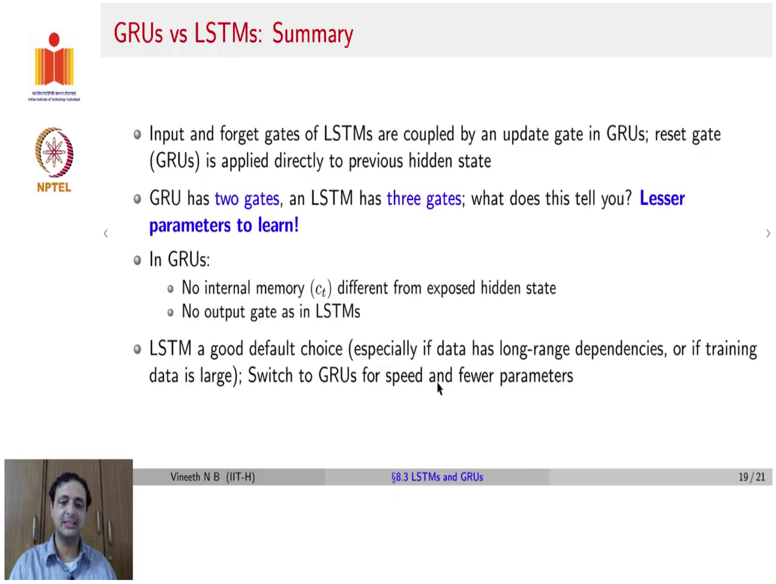
But now the new hidden state content is given by the reset gate, the reset gate looks at h_{t-1}, which is the hidden state coming from the previous time step decides how much of that should be processed and then gives you an updated hidden state. So, the reset gate selects useful parts of the previous hidden state and uses this to compute the new hidden state and the final hidden state that is exposed out of the GRU block is $(1 - z_t) * h_{t-1} + z_t * \overline{h}_t$.

So, the update gate controls what is kept from the previous hidden state and what is updated to the new hidden state. So, to understand this further, let us ask this question. What happens if the reset gate is set to all 1s and update gate is set to all 0s? Let us try to analyse this. If the reset gate is set to all 1s this would just remain h_{t-1} there would be no impact there. It would just remain as h_{t-1}.

If the update gate is all 0s, the second term here would disappear because all those terms would become 0s and the first term here would ensure you will have all 1s which means h_t will simply become h_{t-1} and the second term this entire computation from the current time step would not be considered at all. So, effectively very similar to what we talked about as one of the cases with

LSTMs, here, the same state from the previous time step would be retained and there would be no influence of the current input.

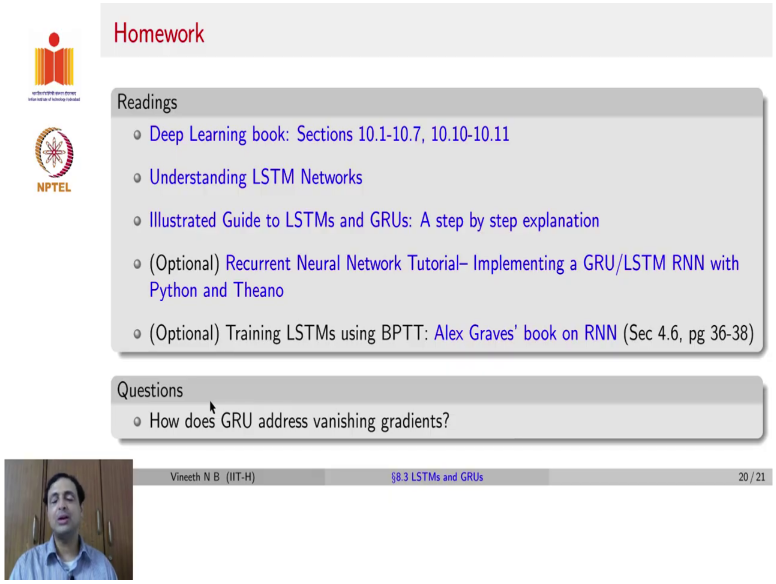
(Refer Slide Time: 26:50)



To summarize the differences between GRUs and LSTMs. The input and forget gates of LSTMs are combined by an update gate in GRUs and the reset gate is applied directly to the previous hidden state in GRUs. So, GRUs have two gates and LSTM has three gates. What does this tell us? Lesser parameters to learn in GRUs. So, learning could be better even with lesser data.

GRUs do not have any internal memory a cell state C_t whatever is the internal cell state is also exposed as it is because there is no formal output gate to control how much of the cell state is output out of the cell state. In general LSTM is a common preferred choice, especially if you know that your data has long range dependencies or if you have large amounts of training data, but if you want better speed and fewer parameters and maybe smaller datasets GRUs are a good choice to try for sequence learning problems.

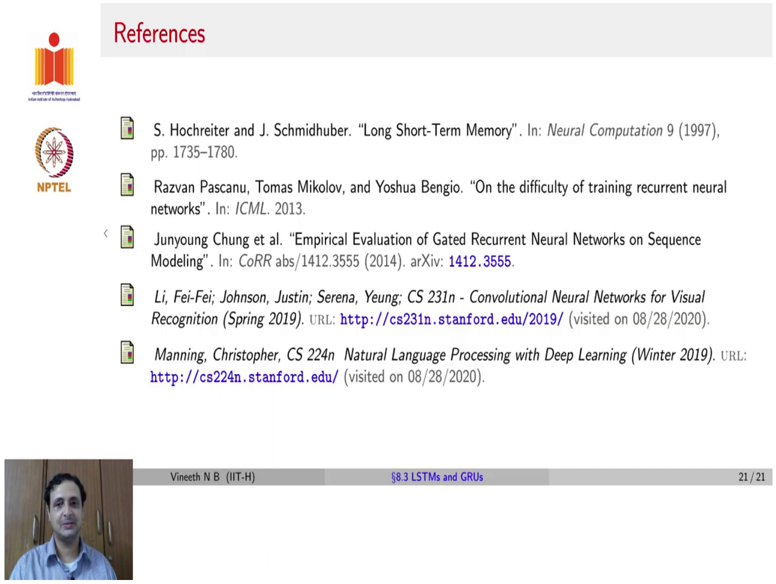
(Refer Slide Time: 28:06)



Your homework would be to continue to read chapter 10 of the deep learning book and these nice links on understanding LSTMs and GRU networks and if you would like to understand how LSTMs are trained using back propagation through time this is Alex Grave's book on RNN, Alex Grave's as we said was responsible for designing the first LSTM and there is also a nice code tutorial here if would like to understand the hands on side. You would have assignments to also get a hands on experience.

But if you like to go through this you can, let us try to leave one question for you to take away. We did answer how LSTMs address vanishing gradients. How do GRUs address vanishing gradients? Try to look at its architecture and think about this question to ponder.

(Refer Slide Time: 29:06)



References.