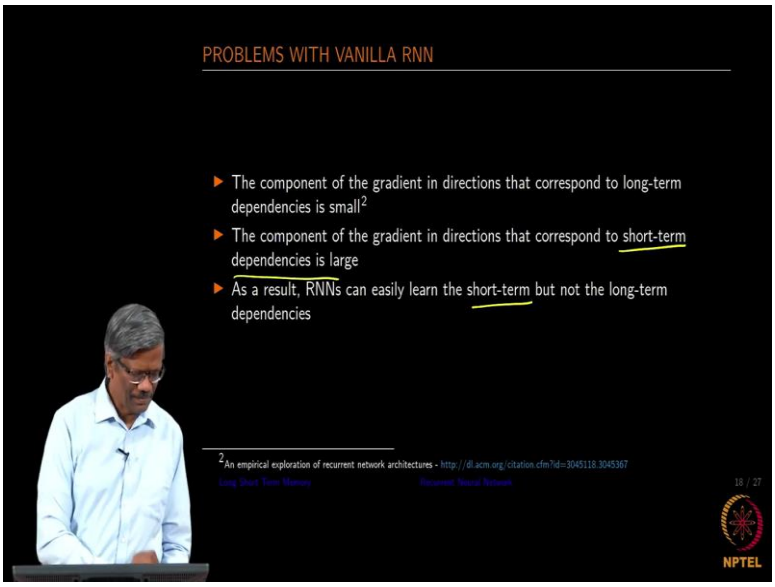
Applied Natural Language Processing
Prof. Ramaseshan Ramachandran
Department of Computer Science and Engineering
Chennai Mathematical Institute, Madras

Lecture – 60
LSTM

(Refer Slide Time: 00:15)

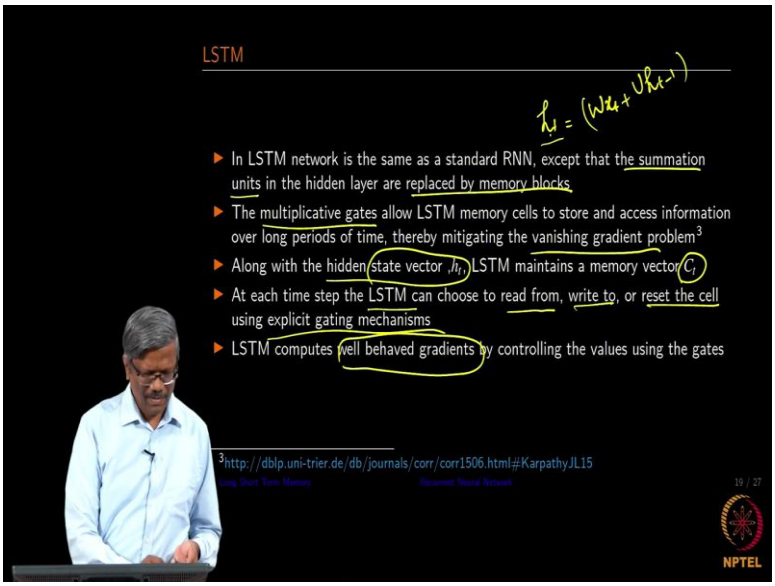


So, as I mentioned earlier what are the problems that we face with the vanilla we call now like the vanilla RNN, because there are going to be variations that are brought into this. So, we call this the vanilla RNN. In the vanilla RNN, the component of the gradient that corresponds to long term dependencies is small, ok. So, as I mentioned earlier when you move from the state t to state 0, the values of the components are becoming smaller and smaller. So, that is the problem of the vanilla RNN. So, but these short-term dependencies are well managed. So, keep a note of this, ok.

So, the long-terms are becoming a problem, but short term dependencies are well managed, ok. So, they can manage the short term better than the long term dependencies, ok. The first assumption that we made that it should be possible for us to really learn a long term dependency is gone with the vanilla RNN. So, people started looking at the options to really correct that, ok. So, this is what I said earlier too, right.

So, when you start using the particular architecture, again and again, you will start finding the problem then we start finding alternatives or make some adjustments to this same network, so that we are able to solve the problem. This is how you know we evolve and the networks evolve as well.

(Refer Slide Time: 02:00)



So, now we are going to be looking at another variation in the RNN. This is called LSTM or Long Short Term Memory, ok. So, where the problem occurs? The problem occurs only in the state where we make the computations, right. When we start making the computations for the state in the unrolled RNN we have a problem. So, is it good enough if you only adjust that particular hidden states, so that we are able to reduce the problem of vanishing gradient? Let us see what LSTM does in that case, ok.

So, in this case, what LSTM has is it instead of one small computation that we make in that s t, you remember that s t, right which is a hyperbolic tangent of h t, correct. So, we are going to be making some changes to that particular computation. We are going to be replacing that summation unit with some memory blocks, ok. So, we are going to have multiple gates that allow cells to keep the information or lose the information, ok.

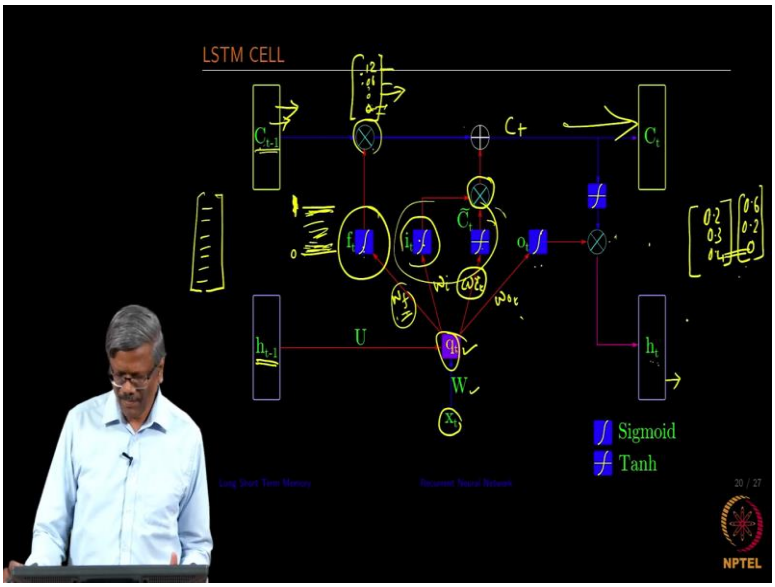
So, by doing so, we want to achieve a long-term dependency on the network. So, that is the idea, right. And also at the same time solve the problem of vanishing gradient, ok. So, in along with the hidden state vector that we have you remember h t, right. So, this is going to replaced by another memory vector called C t. So, this is going to tell us how to

condition the values of u, so that the vanishing gradients problem really disappears in the network.

So, in this case, LSTM can choose to read from write to or reset all these cell values using the gating mechanisms. So, that means, there is a very small computer inside where it is able to read write and then do some reset operations, so that the h values are very well conditioned, ok. So, in my view it really creates a well-behaved gradient, that is what we want to do, right.

So, when you come back from the t state during the backpropagation I want to be able to have a gradient that is very well behaved. So, in my view the intuition is LSTM really computes the gradient beforehand and then keeps it, so that when you come back you really have a gradient to have the smooth backpropagation mechanism, ok. All right.

(Refer Slide Time: 05:18)



So, how does it look? So, it will little more complex than what we saw earlier, ok. For some still RNN is very complex, right and this still adds little more complexity to the RNN mechanism, ok. This is one cell that we are talking about. So, we are going to be replacing this as I wrote earlier, this h t is replaced by the whole this, ok, all right.

So, as I mentioned earlier there is a separate memory vector C t that we have. So, we have calculated assume that we have the previous memory state and then we have the h t minus 1 previous state of the hidden unit. It connects the input through the matrix W or

the weight vector W, let us call it as q t, ok, this is our h t, right. So, since we are going to have something else here I am just calling it as a q t, ok. So, from here we have 1, 2, 3, 4, and 5 different, rather let me put it this way. We have operations here, and then see how these are computed, ok.

Let us first find the q t and then this is our forget gate. I somehow do not understand why this was named as a forget gate. So, this is a sigmoid function it takes the value from the previous memory cell, ok. So, when you come here. So, forget gate is computed by using q t and a weight vector that has here and then there is an input cell that is computed using the weight matrix that you have here this is W I and then we are computing the new memory here we call it as and then this is the output gate we have another, right.

And then each one is connected by either a sigmoid or a hyperbolic tangent function, ok. And then this one is element-wise multiplication, this one, this is element-wise addition, ok, this is element-wise multiplication, ok. Let us see what happens with these. So, when we compute the first q t, we now have to compute the f t. What is f t? It is a sigmoidal function. So, when you do the dot product of q t and Wf, you are going to be making the values stay in between 0 and 1, correct.

So, there will be values which are closer to 0, there will be values which are in between, and there are values closer to 1, right. So, f t is computed using that and then you have translated those values into this form or you have mapped those values of the dot product into 0 and 1. So, this is a vector of this size with some values. We take the value from the previous memory cell assuming that it is computed and then do an element-wise multiplication of this.

So, what happens? When you do an element-wise multiplication supposing if there are let us say this is, ok. So, this becomes 0 here, right. So, it becomes like 012, 0.6 and so on correct, ok. So, what does it mean? So, we are not really asking the machine to forget this value. So, what we are doing is during the training the weights are going be coming through the backpropagation, and we are going to be adjusting in a way that whether the values really correspond to the target that I am looking at or not, that is what we are going to be doing during the training, ok.

That is what is adjusted, right. When you get the error you start propagating the error back and then adjust the weight. What does it mean? That means, I have some error

values to make sure that the value that you are having in the weight is corrected, so that next time when I compute I get the right target.

So, the error mechanism really is making the weights adjusted so that next time it becomes closer to the real target, right. So, in this fashion when we do these values are learned every time. So, when we do the dot product of this and this, we get some values, and then we will have the C t values are this after the element-wise multiplication.

So, 0 means certain elements are not really considered for the training that is what it means. Not every value going away to either 0 or 1 in this case, right some values have retained some values are going to 0. So, if these values which are very close to 0, would be as showing 0s in the vector, correct. So, that means, the memories adjusted in such a way that the values that are coming in through the W and the Wf and through f t either retained or lost, right.

Again and then taking the input gate. So, what we do is we want to find out we want to take the input value as we have done in the previous h t calculation, and then we want to compute the new memory using these, right. And then they both or again multiplied element-wise, and then it is added element-wise to get C t, ok. So, that is a new memory that we are adding. So, this is one small portion that we had earlier. So, now, it is having an f t and then this is computed then added as a new memory and now there is another one which is the output gate, ok.

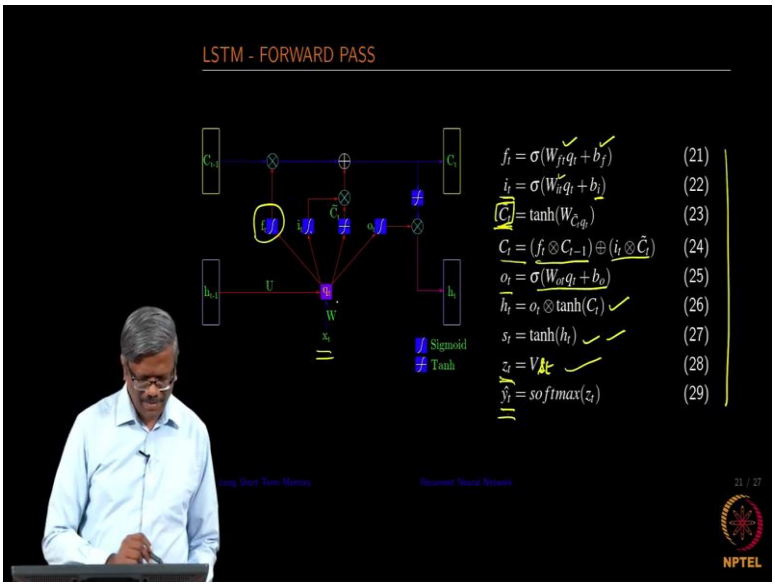
Again, we take the value of q t, use a sigmoid and then whatever value that we had earlier decided that should be passed on, right, as the memory is brought back through the tan h and then the output is combined using you're an element-wise multiplication and finally, the new h t is computed, ok. So, this is during the forward pass, correct.

So, every time you will see that some values appearing and disappearing because of the element-wise addition and multiplication. And then, you also should see that these values are now controlled by these sigmoidal function and the element-wise operation that we are performing. So, the certain thing that we want to retain will continue to be made available through C t and they will be pushed and h t will keep remembering that long-term dependency through the application of C t in the process, ok. Is this clear now, ok.

So, what happens during the backpropagation? Right. So, we have done the forward pass. So, when you do the backpropagation the error is computed, there is a difference with respect to the goal target and that is brought back. When you do that again all the elements that we have completed in the forward pass should be having a derivative and the value is passed on.

So, when you keep doing that, right, the values here in the f t, it, and C t are adjusted by the error mechanism. So, by doing so, by conditioning the matrices because of the error mechanism that we have we seem to be managing to get a good handle on how h t should transfer the values to the next state and so on ok, all right.

(Refer Slide Time: 16:29)



So, this is the forward pass I think I mentioned this in detail there, and these are the equation that we have for the forward pass. First, we use the forget gate and we have the weight and there is a bias. In many cases we ignored bias, LSTM it is important that we manage theme bias as well. Some people would add the bias as part of the s t, so that it does not show up as part of this, ok.

And then we have the input gate that is computed using the q t and the weight that is connecting the input gate and we had a bias to that and then we compute the new memory is there using the hyperbolic tangent function and then we find the new memory. So, now, we compute the new state of the memory using the new memory that

we have computed using this and then we compute the output gate values and then plug in the output gate values along with them a new memory state into h t, ok.

And then, we compute the state of that, and then we compute the this is s t, right output net output, and then we compute the softmax values and then do the backpropagation of this so that all the errors are connected are corrected and we have a stable network.