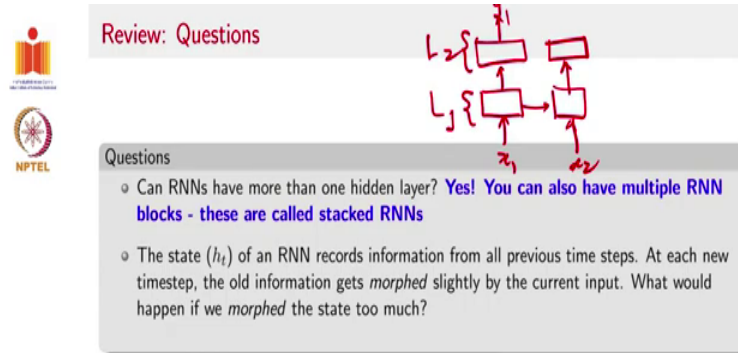


**Deep Learning for Computer Vision**  
**Professor Vineeth n Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture 52**  
**Backpropagation in RNNs**

(Refer Slide Time: 00:22)



The slide features a diagram of two stacked RNN blocks. The bottom block receives an input  $x_1$  and produces an output  $z_1$ . The top block receives the output  $z_1$  as its input and produces an output  $z_2$ . The diagram is annotated with red handwritten text:  $L_2$  and  $L_1$  are written next to the top and bottom blocks respectively, and a bracket groups both blocks with the text "You can also have multiple RNN blocks".

**Review: Questions**

**Questions**

- Can RNNs have more than one hidden layer? **Yes! You can also have multiple RNN blocks - these are called stacked RNNs**
- The state ( $h_t$ ) of an RNN records information from all previous time steps. At each new timestep, the old information gets *morphed* slightly by the current input. What would happen if we *morphed* the state too much?



Vineeth N B. (IIT-H)

§8.2 Backprop in RNNs

2 / 13

We will now move on to Backpropagation in RNNs. Before we go there, we left behind a couple of questions. Can RNNs have more than one hidden layer? We already answered the question. We said that you can have as many hidden layers as you want in each RNN block. In fact, you could also stack RNN blocks if you like, one on top of each other. See you could have an input that goes to one RNN block, whose output goes to another RNN block which is then given to the output at that particular time step.

And then similarly, this RNN block would go over time. And its outputs would go to the upper RNN block. So in such an architecture, which is also known as a stacked RNN, the weights at each level are all shared. So at this level, all the weights are the same across all the time steps. And at level 2, all the weights are the same across all of the time steps. Such an architecture is known as a stacked RNN.

Going forward we asked the question, given that the state of an RNN records information from all previous time steps, what would happen if we morph the state at a given time too much with the current input?

(Refer Slide Time: 01:57)



### Review: Questions

#### Questions

- Can RNNs have more than one hidden layer? **Yes! You can also have multiple RNN blocks - these are called stacked RNNs**
- The state ( $h_t$ ) of an RNN records information from all previous time steps. At each new timestep, the old information gets *morphed* slightly by the current input. What would happen if we *morphed* the state too much? **Effect of previous time-steps will be reduced, may not be desirable for sequence learning problems**



Vineth N B. (IIT-H)

§6.2 Backprop in RNNs

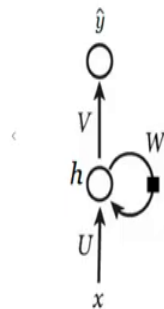
2 / 13

The answer is evident here again, the effect of previous time steps will be reduced, which may not be desirable for sequence learning problems.

(Refer Slide Time: 02:10)



### RNNs: Forward Pass



- Forward pass equations:

$$h_t = \tanh(Ux_t + Wh_{t-1})$$

$$\hat{y}_t = \text{softmax}(Vh_t)$$

- Loss function e.g., Cross Entropy loss:

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) \\ = - \sum_t y_t \log \hat{y}_t$$



Vineth N B. (IIT-H)


§6.2 Backprop in RNNs

3 / 13

Moving on now to backpropagation in RNNs let us first revisit the forward pass in RNNs assuming that this is now your diagram for visualizing an RNN you have an input  $x$  weights  $U$  hidden state  $h$  weights  $W$  then weights  $V$  that take you to an output  $\hat{y}$ , then your forward pass equations are given by  $h_t = \tanh(Ux_t + Wh_{t-1})$ .

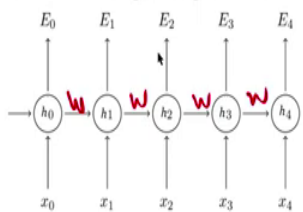
And  $\hat{y}_t = \text{softmax}(Vh_t)$ . This would be your forward pass equations. For an RNN that solving a classification problem where you have the output layer defined by a softmax. What is the cross entropy loss in this setting, you could have used, because it is a classification problem, you could have the standard cross entropy loss as given by this formula.

(Refer Slide Time: 03:20)



### Backpropagation: How?

- **Goal:** Calculate gradients of error  $E$  w.r.t. weights  $U, V, W$
- These gradients will be used to learn weights using SGD; how?



- **Backpropagation Through Time (BPTT):** We sum up gradients at each time step for one training example:  $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$

Credit: Denny Britz, WildML RNN Tutorial

Vineth N B. (IT-H)
§6.2 Backprop in RNNs
4 / 13

Now, if we want to compute the gradients of error  $E$  with respect to the 3 sets of weights that we have here;  $U$ ,  $V$ , and  $W$ , let us assume that these gradients are going to be used to update the weights using stochastic gradient descent exactly the same way we did this for feed forward neural networks, or CNNs.


And it is also important to keep in mind that depending on the kind of RNN variant that you are using, you could have an error in each time step. If you had a many to one setting, you may have an error only at one time step. But in a more general case of an RNN, you could have an error for your output at every time step. So you could have an error  $E_0$  at time step  $t = 0$ . Similarly,  $E_1$  at time step  $t = 1$ , and so on and so forth, in this case till  $E_4$ .

The question now is, how do you compute the gradient of the error with respect to  $U$ ,  $V$  and  $W$ ? How do you do this? It is similar to the general principle of computing the gradient for any other neural network. If a weight influenced an output through multiple paths, then you have to sum up

the contribution of that weight to the output along all possible paths. In our case, you would have a weight here, here, here, here for all the time steps.

And all of them are the same weights in an RNN. So if we had to compute  $\frac{\partial E}{\partial W}$ , where E is an overall error,  $\frac{\partial E}{\partial W}$  would be given by  $\sum_t \frac{\partial E_t}{\partial W}$  where  $E_t$  is the error at each time step. So our next question boils down to how do you compute each of these  $\frac{\partial E_t}{\partial W}$ . Let us see that now.

(Refer Slide Time: 05:35)

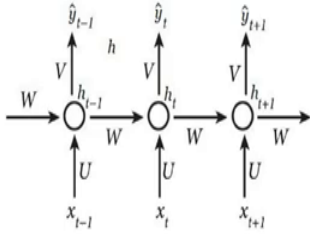


### Backpropagation Through Time (BPTT)


- Consider error at one time step:  $E_3$ ; let us calculate the gradient  $\partial E_3 / \partial V$
- Writing  $z_3 = Vh_3$ , gradient can be calculated as:

$$\begin{aligned} \frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V} \\ &= \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial V} \\ &= (\hat{y}_3 - y_3) \otimes h_3 \end{aligned}$$

where  $\otimes$  is outer product



Credit: Denny Britz, *WildML RNN Tutorial*



Vineth N B. (IIT-H)
582 Backprop in RNNs
5 / 13

Before we go into computing  $\frac{\partial E_t}{\partial W}$  let us take a simpler case, and try to compute  $\frac{\partial E_t}{\partial V}$ . In particular, let us consider  $\frac{\partial E_3}{\partial V}$ , which is, let us say the third time step. So to compute  $\frac{\partial E_3}{\partial V}$ , let us assume that we can write  $z_3$  to be  $Vh_3$  then the gradient can be computed as  $\frac{\partial E_3}{\partial V}$  will be  $\frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V}$ . Now  $\hat{y}_3$  is a softmax of  $z_3$ . That is the way we have defined this network.

So you would have this by chain rule as  $\frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial V}$ . Now, this assuming that you have a linear activation function, or let us assume that this activation function is trivial, and let us assume that  $\frac{\partial E_3}{\partial \hat{y}_3}$ , if you use mean squared error or cross entropy, let us assume that it boils down to a simple

$\widehat{y}_3 - y_3$ , where  $\widehat{y}_3$  is the predicted output and  $y_3$  is the expected output and  $\frac{\partial z_3}{\partial V}$  would be  $h_3$ , because of the definition of  $z_3$  itself.

This becomes the gradient for  $\frac{\partial E_3}{\partial V}$  so you would sum up  $\frac{\partial E_3}{\partial V} + \frac{\partial E_2}{\partial V} + \frac{\partial E_1}{\partial V}$  and so on and so forth to get the gradient of the overall error with respect to  $V$ . Once you compute that, you can update all the weights in  $V$  using gradient descent.

(Refer Slide Time: 07:39)

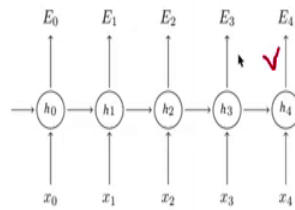
The slide is titled "Backpropagation Through Time (BPTT)". It features a diagram of a recurrent neural network with five hidden states  $h_0, h_1, h_2, h_3, h_4$  and five input-output pairs  $(x_0, E_0), (x_1, E_1), (x_2, E_2), (x_3, E_3), (x_4, E_4)$ . The weight  $W$  connecting  $h_2$  to  $h_3$  is highlighted with a red circle. To the right of the diagram, there are two bullet points: "How about  $\partial E_3 / \partial W$ ?" and "Can we write it as:". Below these is the equation 
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \widehat{y}_3} \frac{\partial \widehat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial W}$$

Now, let us move on to the next case, which is  $\frac{\partial E_3}{\partial W}$ . Recall again that in RNNs we have  $U, V$  and  $W$ . We need to compute the gradients of the error with respect to each of them. So, let us say we have to compute  $\frac{\partial E_3}{\partial W}$ , that would now be written as very similar to what we wrote for  $V$ ,  $\frac{\partial E_3}{\partial W}$  would be  $\frac{\partial E_3}{\partial \widehat{y}_3} \frac{\partial \widehat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial W}$  which is the  $W$  that you have coming into it from the previous layer. The question now is, is this good enough? If we now took this quantity, and summed up  $\frac{\partial E_3}{\partial W} + \frac{\partial E_2}{\partial W}$  and so on. Would we have solved  $\frac{\partial E_3}{\partial W}$  overall?

(Refer Slide Time: 08:42)



### Backpropagation Through Time (BPTT)



- How about  $\partial E_3 / \partial W$ ?
- Can we write it as:

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial W}$$

- It's not complete, since  $h_3$  depends on  $W$ .

$$h_3 = \tanh(Ux_2 + Wh_2)$$

- Chain rule needs to be applied again!

Credit: Denny Britz, WildML RNN Tutorial



Vineth N B. (IIT-H)

§8.2 Backprop in RNNs

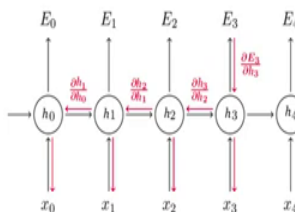
6 / 13

Unfortunately, no, because while  $\partial h_3$  depends on  $W$ ,  $\partial h_3$  also depends on  $h_2$ , which in turn depends on  $W$  again, which means chain rule needs to be applied again to be able to complete this computation of  $\frac{\partial E_3}{\partial W}$ . Why did we not need this with  $V$ ? Because we did not have this problem because it was directly connecting  $h$  to the error. So, how do we complete this?

(Refer Slide Time: 09:19)



### Backpropagation Through Time (BPTT)



- Observe that  $h_3$  depends on  $W$  directly as well as indirectly via  $h_2, h_1, \dots$  hence:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$

- How about  $\partial E_3 / \partial U$ ? Similar to  $\partial E_3 / \partial W$  - Homework!

$$\left( \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \right) \frac{\partial h_3}{\partial W} + \dots \left( \frac{\partial h_1}{\partial h_2} \frac{\partial h_2}{\partial W} \right)$$

Credit: Denny Britz, WildML RNN Tutorial



Vineth N B. (IIT-H)


§8.2 Backprop in RNNs

7 / 13

So,  $h_3$  depends on  $W$  via  $h_2, h_1$  and all other earlier hidden states. So which means  $\frac{\partial E_3}{\partial W}$  can be written as;  $\sum_{k=0}^3 \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$ . What about  $\frac{\partial E_3}{\partial U}$ ? We are going to leave that as homework because it is going to be very similar to  $\frac{\partial E_3}{\partial W}$ , you only have to apply the chain rule in a principled manner. Just to complete this discussion, so, if one had to look at this, how would it look in expansion?


It would look like  $\left(\frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3}\right) \frac{\partial h_3}{\partial W} + \left(\frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3}\right) \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W}$  and you will have further summations that do similar chain rules for  $h_1$  and so on and so forth. Remember, this summation now is only for  $\frac{\partial E_3}{\partial W}$ , you will have, similarly another summation for  $\frac{\partial E_4}{\partial W} \frac{\partial E_2}{\partial W}$  so on and so forth. And your final gradient for  $W$  has to add up all of those to compute  $\frac{\partial E}{\partial W}$ .

(Refer Slide Time: 11:20)



### Backpropagation Through Time (BPTT)

- Observe that  $\frac{\partial h_3}{\partial h_k}$ , when  $k = 1$ , will be further expanded, using chain rule, as:
 
$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$
- Consequently, gradient  $\frac{\partial E_3}{\partial W}$  can be written as:
 
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \left( \prod_{j=k+1}^3 \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$



Vineeth N B. (IIT-H)
16.2 Backprop in RNNs
8 / 13

So, if you now observe  $\frac{\partial h_3}{\partial h_k}$  when  $k = 1$  as we just said, can be expanded as  $\frac{\partial h_3}{\partial h_1}$  would be  $\frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$ . So, this entire gradient can now be succinctly written as  $\sum_{k=0}^3 \frac{\partial E_3}{\partial \widehat{y_3}} \frac{\partial \widehat{y_3}}{\partial h_3} \left( \prod_{j=k+1}^3 \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$ .

(Refer Slide Time: 12:08)



### Backpropagation Through Time (BPTT)

Do you see any problem?

Sequences (sentences) can be quite long, perhaps 20 words or more - need to backpropagate through many layers!  $\Rightarrow$  **Vanishing Gradient Problem!**



Vineth N B. (IIT-H)

§8.2 Backprop in RNNs

9 / 13



### Backpropagation Through Time (BPTT)

- Observe that  $\partial h_3 / \partial h_k$ , when  $k = 1$ , will be further expanded, using chain rule, as:

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

- Consequently, gradient  $\partial E_3 / \partial W$  can be written as:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \left( \prod_{j=k+1}^3 \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$



Vineth N B. (IIT-H)

§8.2 Backprop in RNNs

8 / 13


Do you see any problem in this particular approach? If you thought carefully, you will realize that RNNs are often used for time series data that can be reasonably long. You could be using it for data that has 20 time steps, 50 time steps, 100 time steps depending on the nature of the problem that you are dealing with.

So when you now back propagate, you are going to be multiplying the gradients across all of these time steps. So if you saw in the slide earlier, you would have this term which continues to multiply these activations across multiple time steps. Now, why could that cause a problem? If your gradient for each of those values is less than 1, multiplying these terms will lead to a

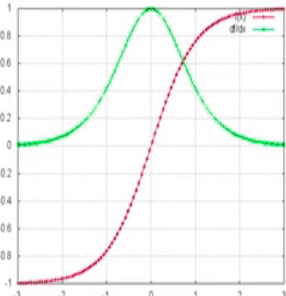


vanishing gradient problem, because the multiplication of values less than 1 will quickly go to 0. Is this really a problem?

(Refer Slide Time: 13:15)




### Vanishing Gradient Problem



- Observe the equation:


$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left( \prod_{j=k+1}^3 \frac{\partial h_j}{h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

- For sigmoid activations, gradient is upper bounded by 1; what does this tell us?
- Gradients will vanish over time, and long-range dependencies will not contribute at all! How to combat?

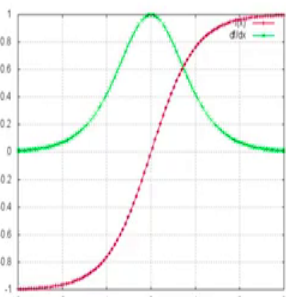


Vineeth N B. (IIT-H) §8.2 Backprop in RNNs

10 / 13




### Vanishing Gradient Problem



- Observe the equation:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left( \prod_{j=k+1}^3 \frac{\partial h_j}{h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

- For sigmoid activations, gradient is upper bounded by 1; what does this tell us?
- Gradients will vanish over time, and long-range dependencies will not contribute at all! How to combat? We'll see in the next lecture



Vineeth N B. (IIT-H) §8.2 Backprop in RNNs

10 / 13

Credit: Denny Britz, WildML RNN Tutorial

Let us consider, say, a sigmoid activation function that we use in a layer in the RNN. So we know that the sigmoid function is upper bounded by 1, the values lie between 0 and 1. Let us, even if we took a tanh activation function, it would lie between -1 and 1. So the gradient of the sigmoid activation function, it is also upper bounded by 1 which means all these terms will have gradients which are upper bounded by 1.

And what does that tell us? It means that the gradients in this particular computation  $\frac{\partial E_3}{\partial W}$  will quickly vanish over time. And an earlier time step. The weights or the impact of an earlier time step may never be felt on a later time step. Because the gradients that you get due to an earlier time step, it is most likely will become 0 because of this product over a long range of activations across many time steps.

So effectively, although you want RNNs to model long term temporal relations, you may not really be able to achieve that purpose because of the vanishing gradient problem, because an earlier time step may not really influence an output at a later time step. How do you combat this problem? We will see this in the next lecture. There are already solutions for this problem. And we will see this in the next lecture.

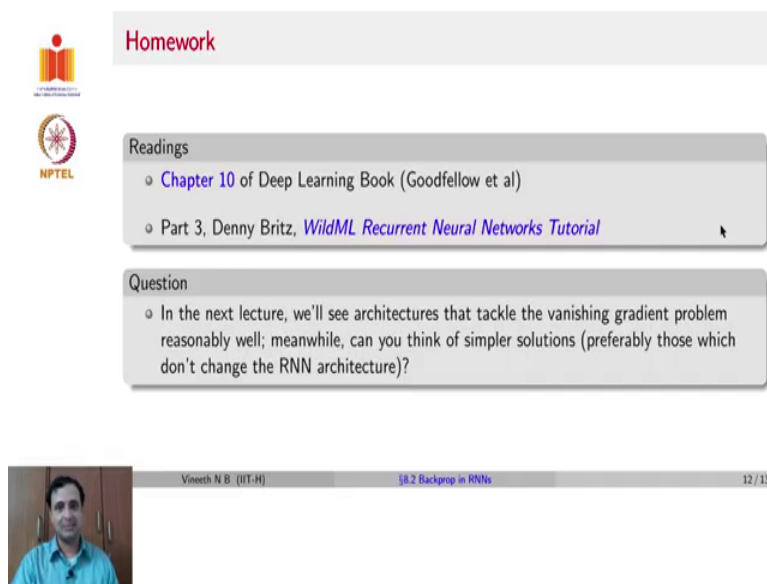
(Refer Slide Time: 14:56)

But before we go there let us ask the counter question. What if I did not use a sigmoid activation function? What if I just use the linear activation function let us assume, on the contrary, that each of my gradients  $\frac{\partial h_3}{\partial h_2}, \frac{\partial h_2}{\partial h_1}$  were very high values, then multiplying all of them could lead to what is known as the exploding gradient problem, because the product of values, say in the range of 10, by multiplying 3 such values, you will quickly go to  $10^3$  magnitude.

And that can lead to an explosion,exploding gradient problem. This generally is not too much of an issue during implementation. Can you think why this may be the case? The answer is, firstly, it is likely to show up as NaN, not a number during implementations. And more importantly, you can simply clip the gradients beyond a particular value.

This is known as gradient clipping. And it is very popularly done today, while training neural networks, where you say that if the gradient exceeds 10, you are going to stop the maximum value it can obtain as 10. So even if your gradient was  $10^3$ , you are only going to choose it as 10 and move on with the rest of the computations. This generally takes care of the exploding gradient problem, although the vanishing gradient problem remains, and we will see this in the next lecture.

(Refer Slide Time: 16:40)



The screenshot shows a presentation slide with the following content:

- Homework**
- Readings**
  - Chapter 10 of Deep Learning Book (Goodfellow et al)
  - Part 3, Denny Britz, *WildML Recurrent Neural Networks Tutorial*
- Question**
  - In the next lecture, we'll see architectures that tackle the vanishing gradient problem reasonably well; meanwhile, can you think of simpler solutions (preferably those which don't change the RNN architecture)?

At the bottom of the slide, there is a small video feed of a man in a blue shirt, and a footer containing the name 'Vineeth N B. (IIT-H)', the slide title '§8.2 Backprop in RNNs', and the page number '12 / 13'.

So your homework for this lecture is, continue to read chapter 10 of the Deep Learning book, and also go through this excellent WildML RNN tutorial by Danny Britz, which explains backpropagation and RNN very very well. The question that we are going to leave behind at the end of this lecture is, as I just mentioned, in the next lecture, we will see how you can change an RNN architecture to avoid the vanishing gradient problem.

But can you solve or address the vanishing gradient problem, without any change in the overall architecture? Through some choices, can you solve the vanishing gradient problem? Think about it and we will discuss the next time.

(Refer Slide Time: 17:33)



## References

- Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), 1735–1780.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *ICML*. 2013.
- Kyunghyun Cho et al. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *SSST@EMNLP*. 2014.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.



Vineth N.B. (IIT-H)

§8.2 Backprop in RNNs

13 / 13

The references.