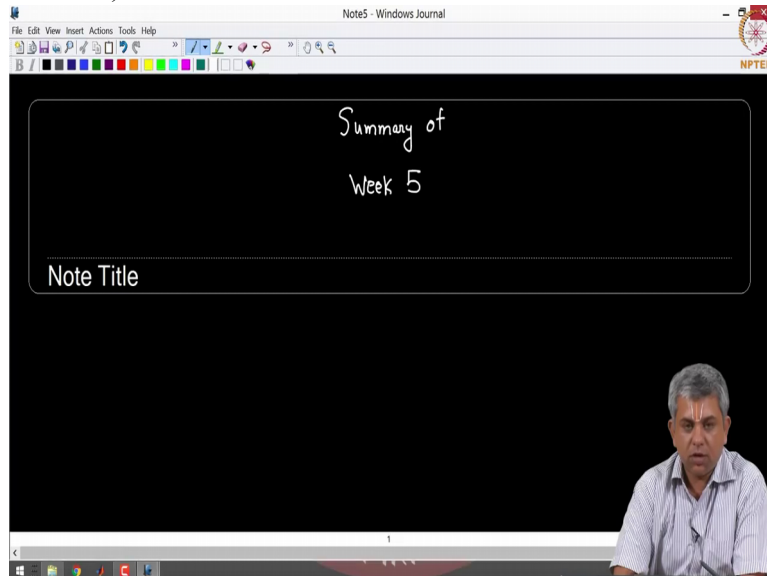


Machine Learning for Engineering and Science Applications
Professor Doctor Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology Madras
Summary of Week 5

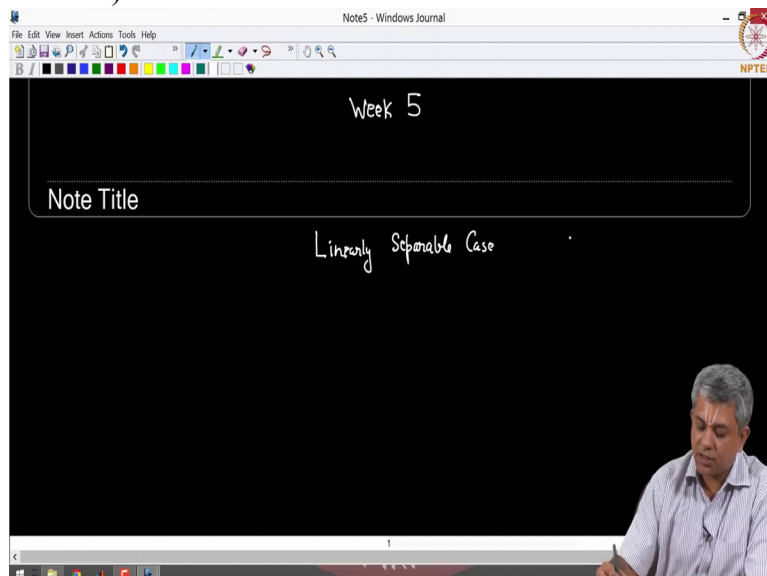
(Refer Slide Time: 00:13)



In this video we will just summarize what all we have seen in week 5 and also what we did not see in week 5. And just to give you a preview of next week what we will be doing.

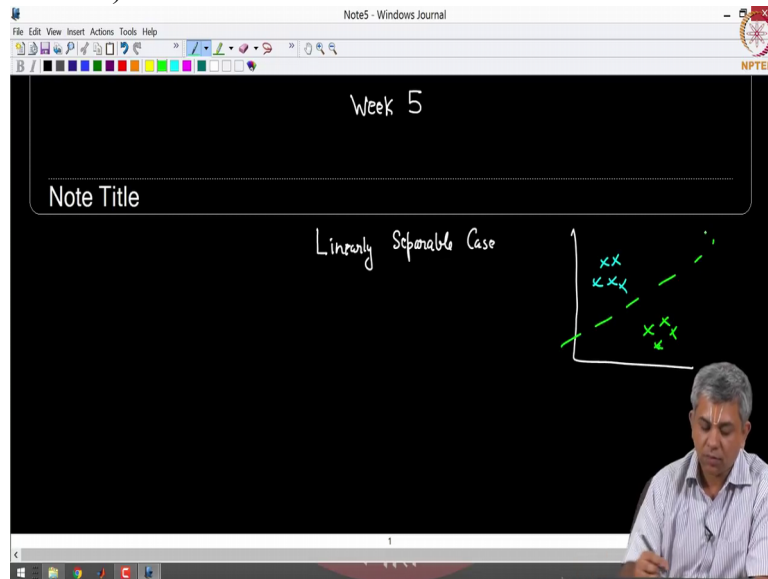
So what we saw was when you have a linearly separable classification case

(Refer Slide Time: 00:37)



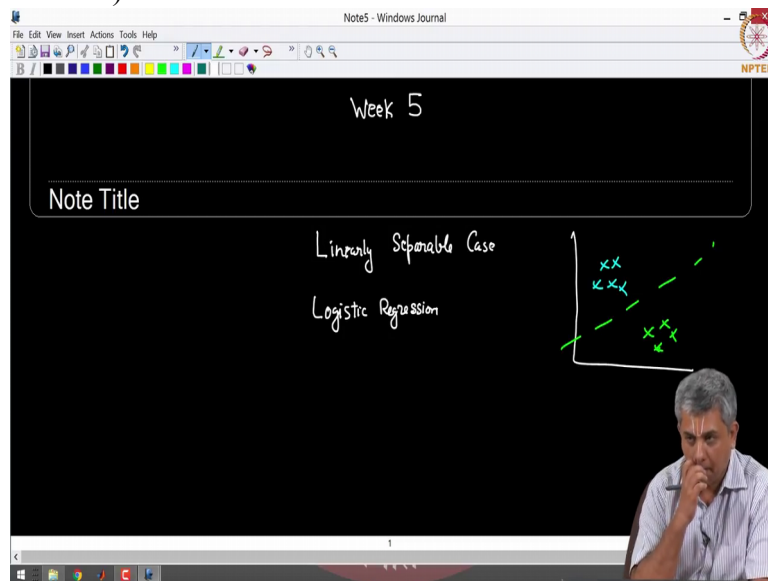
that is if you have data points which can simply be separated by a line such as this data set.

(Refer Slide Time: 00:48)



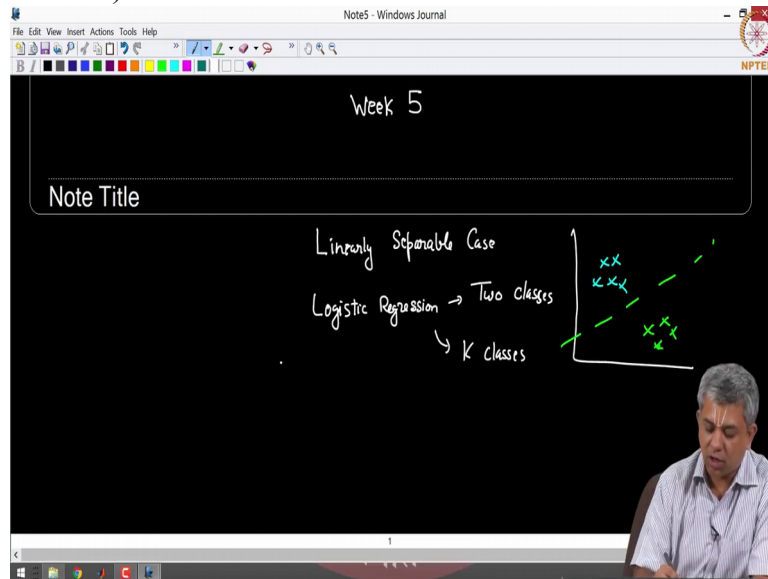
In such a case you could use logistic regression.

(Refer Slide Time: 00:58)



Logistic regression or binary logistic regression can be used when there are just 2 classes. And the same idea we saw could be extended to k classes

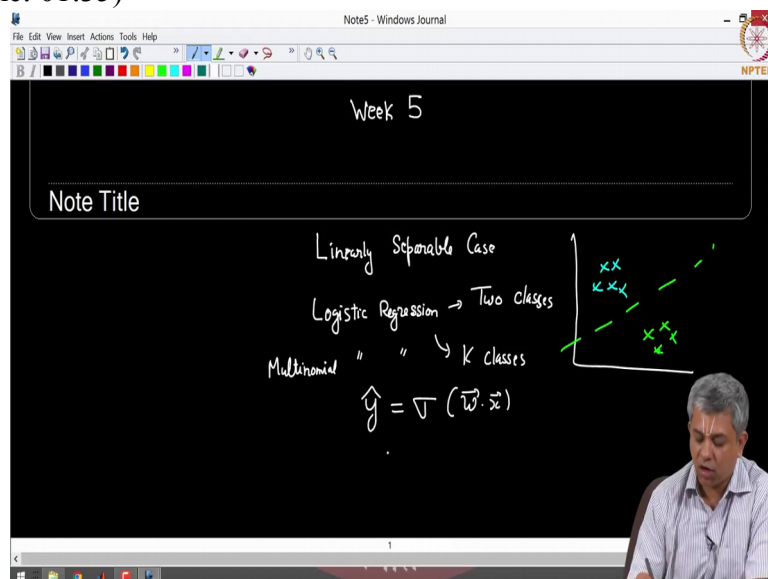
(Refer Slide Time: 01:14)



using multinomial logistic equation.

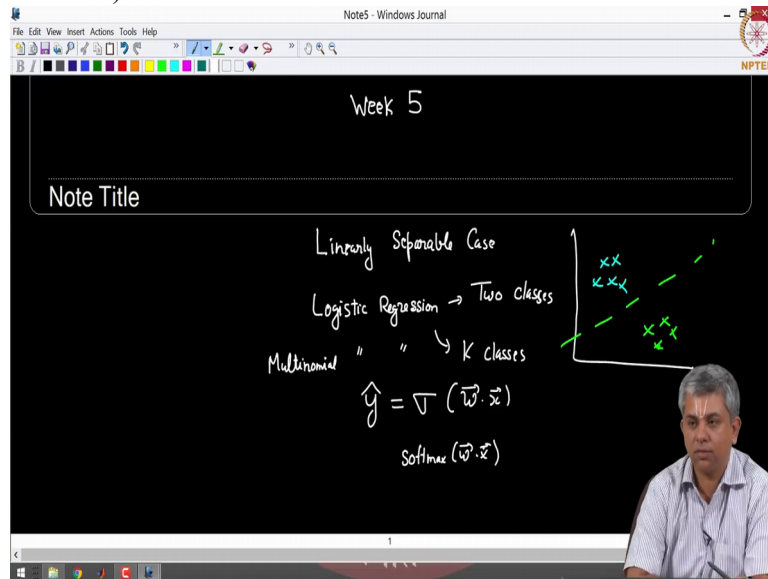
In both these cases the major differences were simply in the forward model. The forward model for logistic regression was sigmoid of $w \cdot x$. And

(Refer Slide Time: 01:35)



for multinomial logistic regression was Softmax of $w \cdot x$.

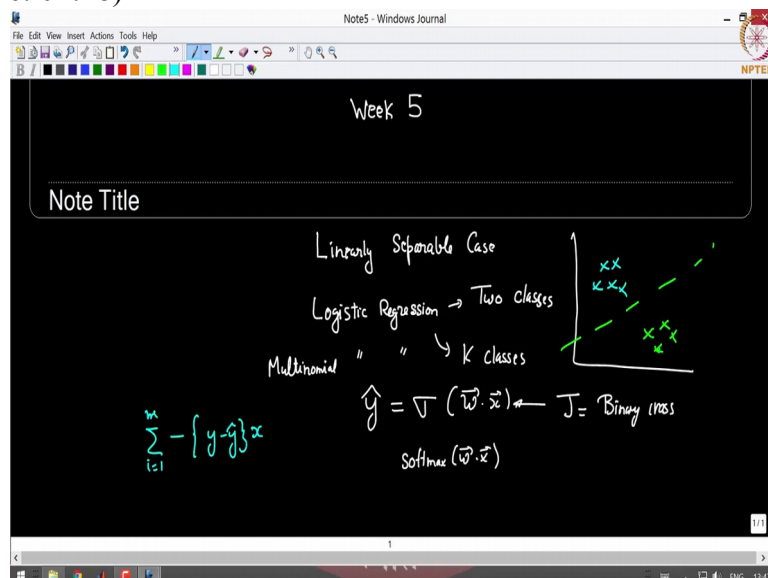
(Refer Slide Time: 01:42)



Now apart from this we also had our loss function which was the binary class entropy loss function for logistic regression. And in the case of multinomial we saw that it was a simple extension. It was a general cross entropy loss function there.

In both these cases it was fairly straight forward in calculating del J del w. It turned out to give us the same expression as before which was y minus y hat times x summation from i equal to 1 to m.

(Refer Slide Time: 02:18)



Now this followed our general machine learning paradigm which is, you take x, guess a w, get a y hat back propagate. This is what we did in logistic as well as multinomial logistic

regression cases. Therefore these, when we tried this for XOR we saw that it needed an extra layer in the middle.

(Refer Slide Time: 02:45)

Linearly Separable Case

Logistic Regression \rightarrow Two classes

Multinomial " " \rightarrow k classes

$\hat{y} = \sigma(\vec{w} \cdot \vec{x}) \leftarrow J = \text{Binary Cross}$

Softmax $(\vec{w} \cdot \vec{x})$

XOR \rightarrow Extra Layer.

$\sum_{i=1}^m -\{y_i \log \hat{y}_i\}$

It is not possible to simply take an input and map it directly to an output

(Refer Slide Time: 02:50)

Linearly Separable Case

Logistic Regression \rightarrow Two classes

Multinomial " " \rightarrow k classes

$\hat{y} = \sigma(\vec{w} \cdot \vec{x}) \leftarrow J = \text{Binary Cross}$

Softmax $(\vec{w} \cdot \vec{x})$

XOR \rightarrow Extra Layer.

$\sum_{i=1}^m -\{y_i \log \hat{y}_i\}$

$\text{X} \rightarrow \text{1}$

without any hidden layer.

However with the extra layer it is possible, it can be proved that you have universal approximation here which says

(Refer Slide Time: 03:04)

Linearly Separable Case

Logistic Regression \rightarrow Two classes

Multinomial " " \rightarrow K classes

$\hat{y} = \nabla (\bar{w} \cdot \bar{x}) \leftarrow J = \text{Binary Cross Entropy}$

Softmax $(\bar{w} \cdot \bar{x})$

XOR \rightarrow Extra Layer $\odot \rightarrow \oplus$

Universal Approx. Thm.

that any function can actually be approximated to an arbitrary degree of accuracy provided you are willing to increase your number of neurons. It is possible to approximate any function to an arbitrary degree of accuracy using one single hidden layer.

Neural networks however use more than one hidden layer and there is some disagreement on, in the literature on this.

(Refer Slide Time: 03:33)

Multinomial " " \rightarrow K classes

$\hat{y} = \nabla (\bar{w} \cdot \bar{x}) \leftarrow J = \text{Binary Cross Entropy}$

Softmax $(\bar{w} \cdot \bar{x})$

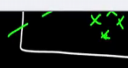
XOR \rightarrow Extra Layer $\odot \rightarrow \oplus$

Universal Approx. Thm.


Neural Networks

More than one hidden layer and this is what is called

(Refer Slide Time: 03:43)

Multinomial " " \rightarrow K classes 

$$J = \text{Binary Cross Entropy}$$
$$\text{Softmax}(\vec{w} \cdot \vec{x})$$

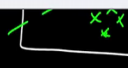
XOR \rightarrow Extra Layer 

Universal Approx Thm.


Neural Networks \rightarrow More than one hidden layer

deep learning. Deep learning simply means greater than one hidden layer.

(Refer Slide Time: 03:51)

Multinomial " " \rightarrow K classes 

$$J = \text{Binary Cross Entropy}$$
$$\text{Softmax}(\vec{w} \cdot \vec{x})$$

XOR \rightarrow Extra Layer 

Universal Approx Thm.

Neural Networks \rightarrow More than one hidden layer

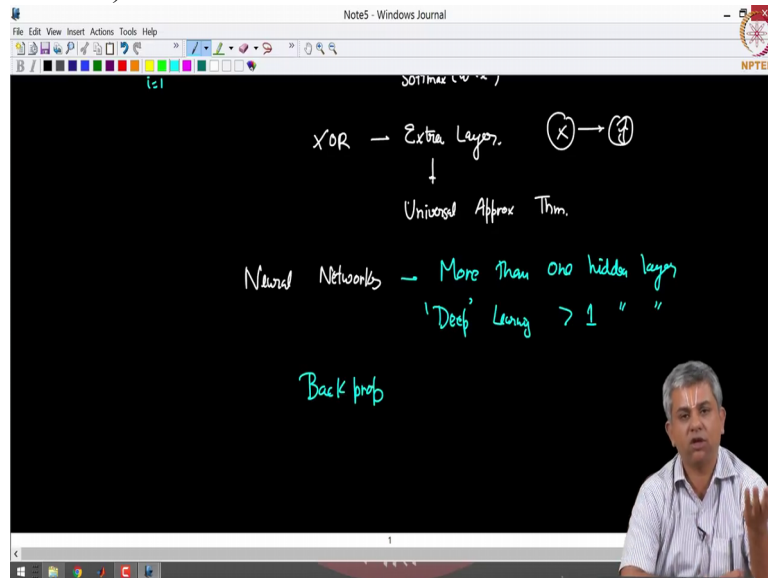
'Deep' Learning > 1

That is typically what is called deep learning. There is some disagreement in the literature on this, on how many layers should you take, or should you even just make do with one hidden layer.

Some people are of the opinion that with certain tricks you can get by, but generally the observation is you get fewer neurons and fewer weights the deeper that you go, Ok.

Now in order to train deep neural network you need however back propagation algorithm of which

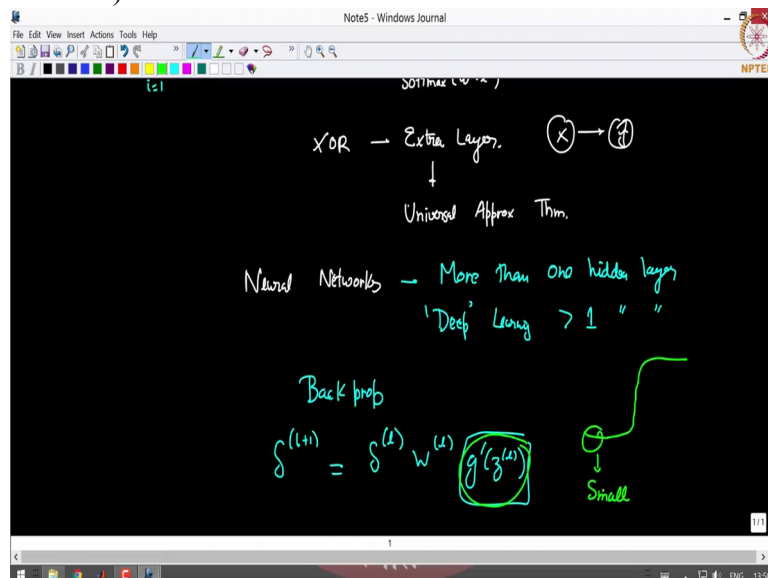
(Refer Slide Time: 04:21)



we saw the rudiments in the previous video. Now one thing that tends to happen is if you recall our expression was δ_{l+1} was δ_l times g' prime z .

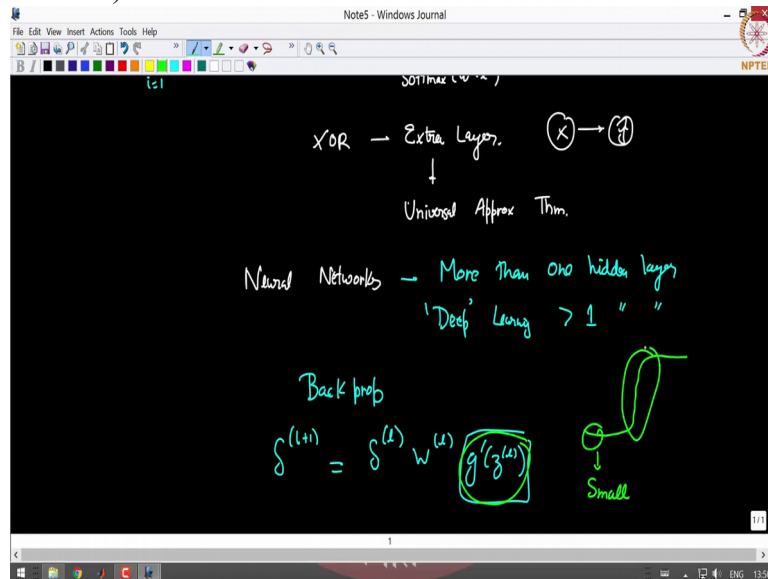
Now notice this term g' prime z . When you have a sigmoid this g' prime or the slope of the sigmoid can actually get small,

(Refer Slide Time: 04:55)



further and further away from this central portion which has

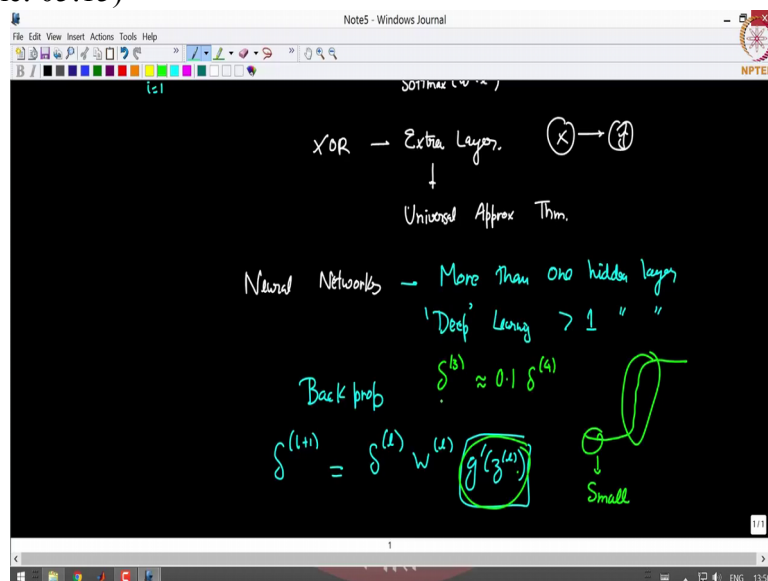
(Refer Slide Time: 04:59)



high slope. Further and further away you get, this can get very small. And it can keep on multiplying.

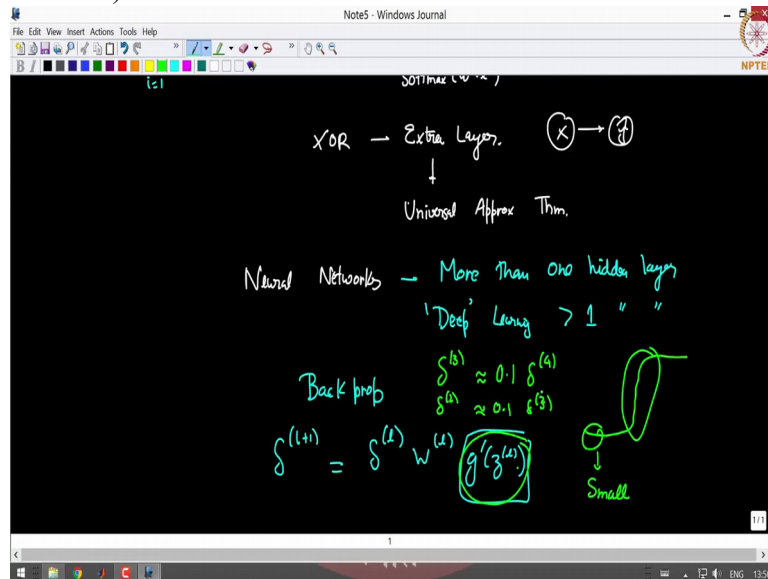
So you have delta 3 is some small number, let us say point 1 multiplying delta 4.

(Refer Slide Time: 05:15)



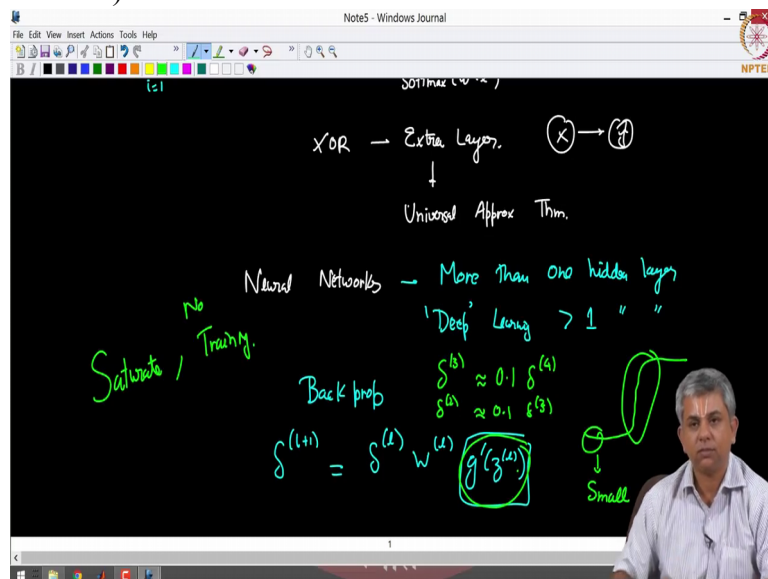
delta 2 will be that small number multiplying delta 3, so on and so forth.

(Refer Slide Time: 05:21)



So if the small numbers keep on multiplying, it can actually get very, very small and it can go below the machine epsilon and then network will, what is called, it will not train.

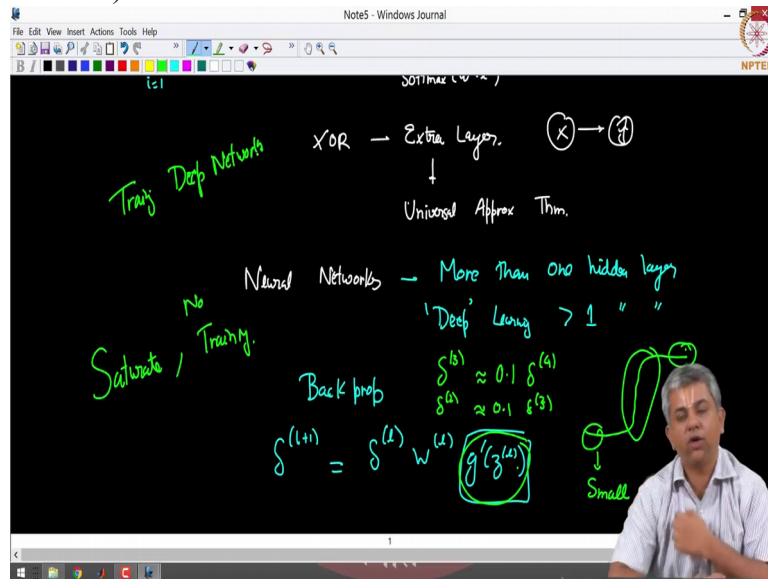
(Refer Slide Time: 05:39)



Similarly here too it will stop training. This is called saturation, that is your value is so close that your slopes are very, very low. And this is the problem, fundamental problem in training deep networks.

You tend to get one of two

(Refer Slide Time: 06:00)



problems which you will also see in the next few weeks which is either of exploding gradients or of vanishing gradients. That is, w actually completely blows up of which we saw few examples even during linear regression. That was due to improper gradient descent.

Or you could have something which you think should train but it does not train. And this is where a lot of neural research stagnated.

So there are tricks in order to do this and you will see Doctor Ganapathy will discuss several tricks for this in the context of convolutional neural networks next week. What is it that we did not cover?

So one was this. The other things that we did not cover and which we will be looking at next weeks is how do we initialize w ? As we saw

(Refer Slide Time: 06:54)

The screenshot shows a blackboard with handwritten notes in green and white. The notes include:

- Saturated / Training.*
- Deep Learning > 1*
- Back prop*
- $\delta^{(3)} \approx 0.1 \delta^{(4)}$
- $\delta^{(2)} \approx 0.1 \delta^{(3)}$
- $\delta^{(l+1)} = \delta^{(l)} w^{(l)}$ (with $g'(z^{(l)})$ circled)
- A diagram of a neuron with a small output labeled *Small*.
- Did not cover*
- ① How do we initialize \vec{w} ?

even for logistic regression or neural networks the minimum is not unique. Since it is not unique how you initialize actually has the effect on how your neuron network trains.

Second thing is how do we determine the number of layers, number of neurons per layer. I just showed something arbitrary here. Both these are also hyper parameters. Remember

(Refer Slide Time: 07:33)

The screenshot shows a blackboard with handwritten notes in green and white. The notes include:

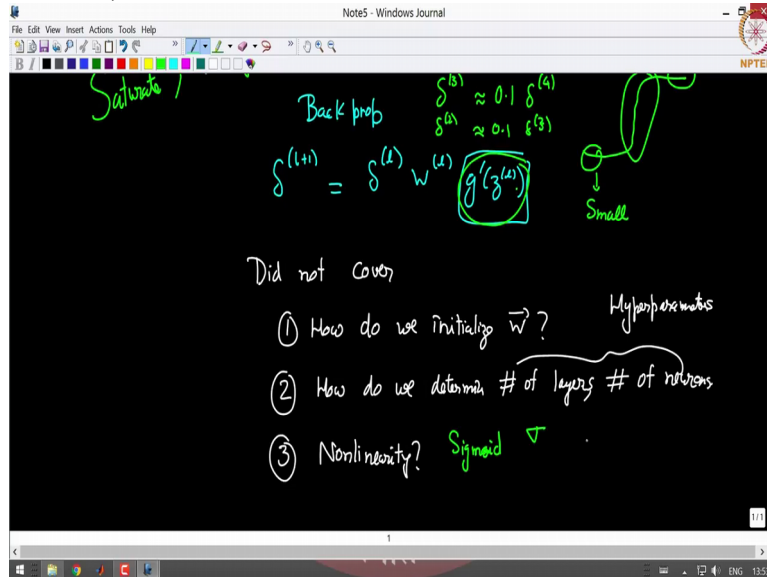
- Saturated / Training.*
- Deep Learning > 1*
- Back prop*
- $\delta^{(3)} \approx 0.1 \delta^{(4)}$
- $\delta^{(2)} \approx 0.1 \delta^{(3)}$
- $\delta^{(l+1)} = \delta^{(l)} w^{(l)}$ (with $g'(z^{(l)})$ circled)
- A diagram of a neuron with a small output labeled *Small*.
- Did not cover*
- ① How do we initialize \vec{w} ?
- ② How do we determine # of layers # of neurons
- Hyperparameters*

in addition to alpha which is your learning rate, and lambda which is your regularization parameter, number of neurons, number of layers per neurons all these are also treated as hyper parameters.

And hyper parameter optimization is a big problem. It is an open problem in neural networks. Doctor Ganapathy will be discussing a few details about this later.

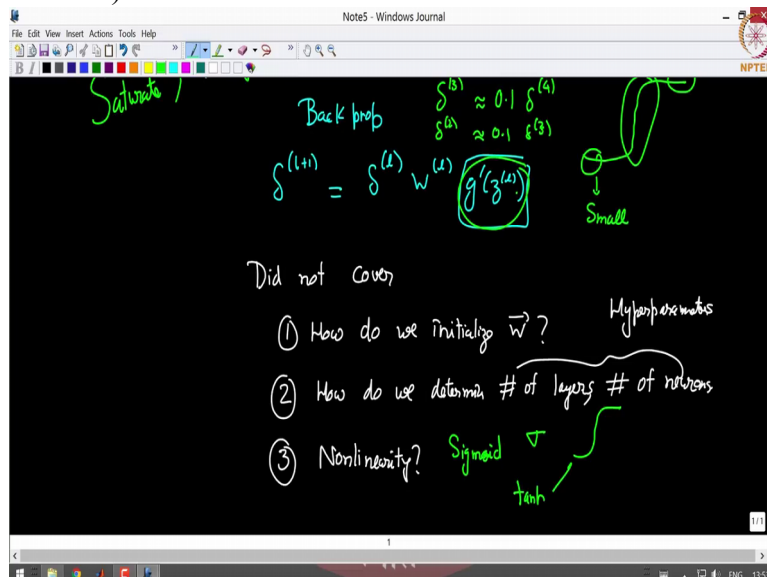
Finally what nonlinearity do you use? I showed just one. I showed sigmoid.

(Refer Slide Time: 08:10)



But there are other possible nonlinearities that people use. One is tan h which is very

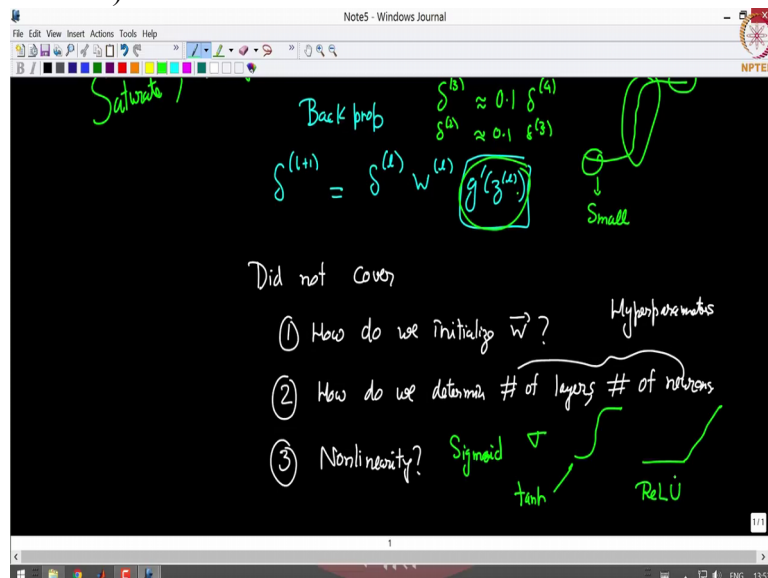
(Refer Slide Time: 08:19)



similar to the sigmoid and instead of going from 0 to 1; it goes from minus 1 to 1.

Another possibility is something called rectified linear unit. In short

(Refer Slide Time: 08:30)



it is called Re L U. It is completely flat at one end and then it is simply linear.

Now different choices can be made for different problems. As a very, very simple rule of thumb, for problems with numbers we tend to use artificial neural networks and we tend to use tan h instead of sigmoid.

For convolutional neural networks we tend to use R L U which you will see in the next week. So these and other issues we will be seeing in the following week. And the final heads up for next week we will be moving to what is called convolutional neural networks also called C N Ns.

They are a special case of A N Ns, or artificial neural networks or deep neural networks that we just saw for vision problems.

(Refer Slide Time: 09:31)

① How do we initialize \vec{w} ? Hypoparameters

② How do we determine # of layers # of neurons

③ Nonlinearity? Sigmoid tanh ReLU

Convolutional Neural Networks (CNNs)

ANNs for vision problems

Is there any problem with ANNs that we cannot use it for vision problems? No, not really.

The only issue is that, let us take my favorite example, that of a 60 cross 60 image. Let us say you have 3600 features, this is just linear features and suppose you have 3600 in the next

(Refer Slide Time: 09:54)

① How do we initialize \vec{w} ? Hypoparameters

② How do we determine # of layers # of neurons

③ Nonlinearity? Sigmoid tanh ReLU

Convolutional Neural Networks (CNNs)

ANNs for vision problems

○ ○
○ ○
⋮ ⋮
○ ○
3600 3600

layer also, so you can see that this is 3600 square weights already which is a huge number of weights.

(Refer Slide Time: 10:02)

The image shows a video lecture interface. The main content is a blackboard with handwritten text and diagrams. At the top, it lists three questions: (1) How do we initialize \vec{w} ? (2) How do we determine # of layers # of neurons? (3) Nonlinearity? To the right of these questions, the word 'Hyperparameters' is written. Below the questions, three activation functions are shown: Sigmoid, tanh, and ReLU. The Sigmoid function is a green curve, tanh is a green S-shaped curve, and ReLU is a green line that is zero until a certain point and then increases linearly. The word 'Linear' is written next to the ReLU graph. Below the blackboard, the text 'Convolutional Neural Networks (CNNs)' is written, followed by 'ANNs for vision problems'. On the left side, there are two columns of circles representing neurons, with the numbers 3600 and 3600 written below them. At the bottom left, the text $(3600)^2$ weights is written. In the bottom right corner, there is a small inset video of a man with grey hair, wearing a light blue shirt, who appears to be speaking.

And vision problems deal with large images so you will have very large features which means you have to deal with a huge number of weight. So instead of doing that the trick is to use what is known as convolutional neural networks. We will start seeing that from next week. Thank you.