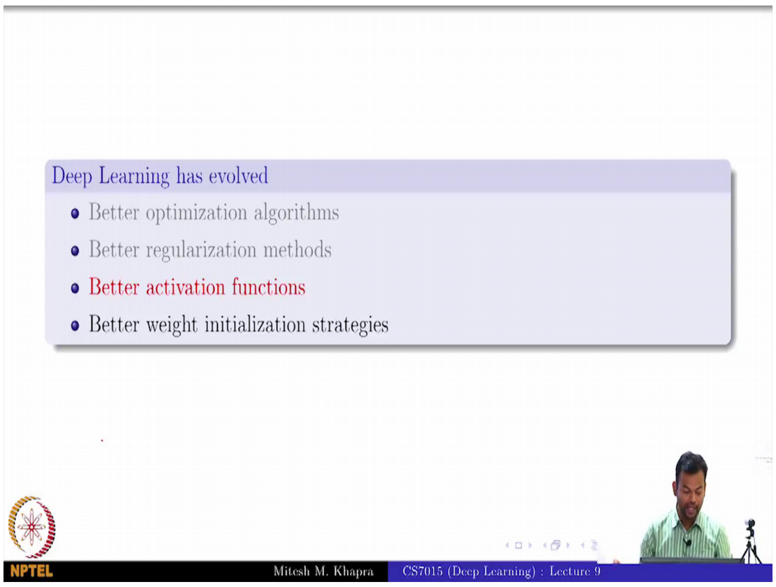
Deep Learning

Prof. Mitesh M Khapra
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 9.3
Lecture – 09
Better activation functions

Let us start with Better activation functions.

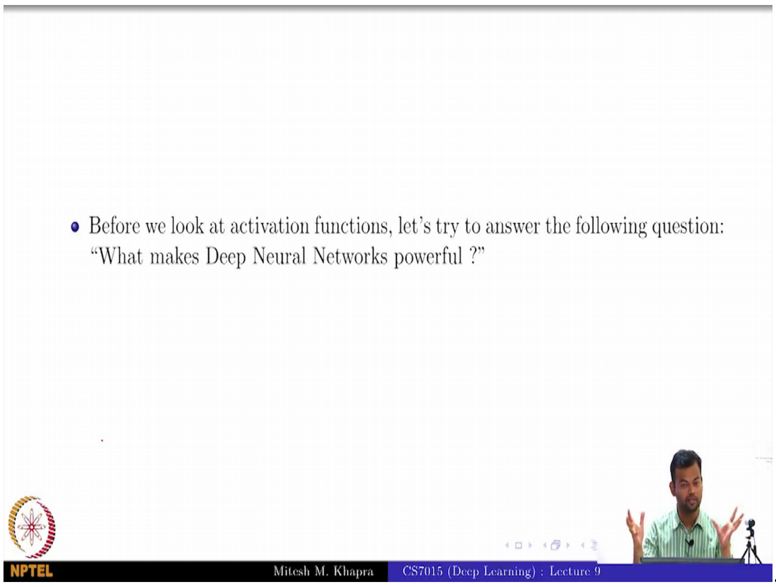
(Refer Slide Time: 00:16)



Deep Learning has evolved

- Better optimization algorithms
- Better regularization methods
- Better activation functions
- Better weight initialization strategies

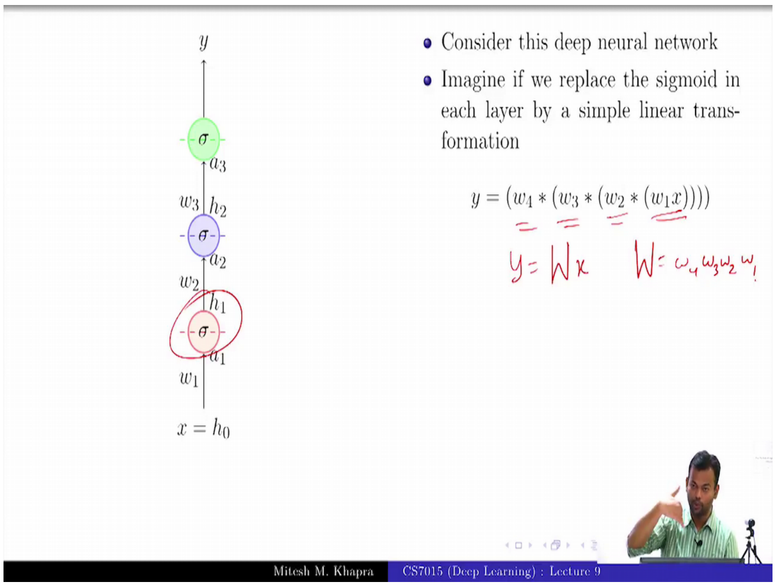
(Refer Slide Time: 00:17)



- Before we look at activation functions, let's try to answer the following question: "What makes Deep Neural Networks powerful ?"

So, before I get into activation functions right, let me first tell you why I care about activation functions, why do I actually want to come up with better activation functions so, will start with the following question. What makes deep neural networks powerful among other things? What is this one thing which makes it powerful? So, let me give you this intuition.

(Refer Slide Time: 00:34)



- Consider this deep neural network
- Imagine if we replace the sigmoid in each layer by a simple linear transformation

$$y = (w_4 * (w_3 * (w_2 * (w_1 x))))$$

$$y = Wx \quad W = w_4 w_3 w_2 w_1$$

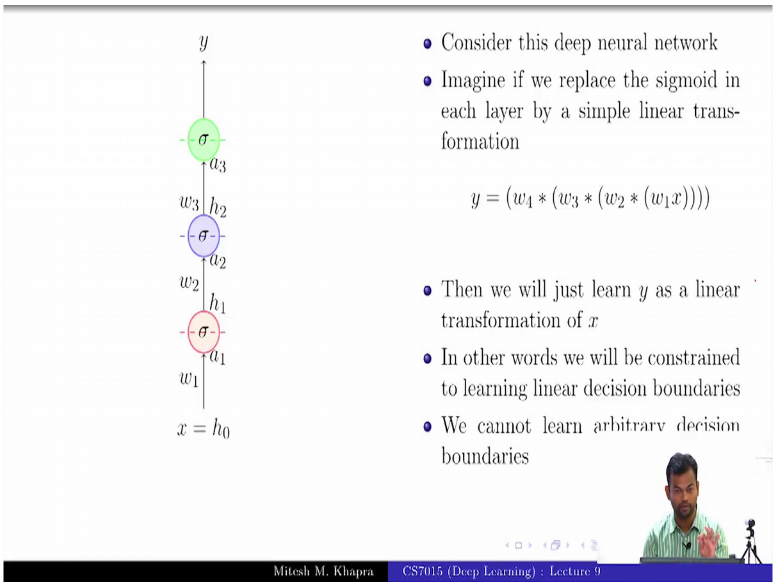
This I have a deep network ok, and do not worry it is a thin network, but I could have had a wide network also, but just for illustration I have taking a deep network thin network

Now imagine that, each of these neurons that you have. If I replace the sigmoid in each layer by a simple linear transformation a by the way this is technically incorrect. So, orange is always input. So, this should not be a sigmoid there right, either add one more layer there or let us change the figure fine.

So, suppose I replace all these sigmoids by linear transformations, what would y be can you write? Y as a function of x. What would it be give me the function? Will we just be this right. So, first we will do w 1 of x which is this right, then will take w 2 of that then w 3 of that and w 4 of that right.

So, I could actually have written this just as y equal to W x where W is equal to w 4 w 3 w 2 w 1. So, there is no depth here, there is actually only one weight which I could have learned you get that right. If you just have all linear transformations, then essentially you do not have so many weights, you just have one weight throughout you get that make sense.

(Refer Slide Time: 01:53)



- Consider this deep neural network
- Imagine if we replace the sigmoid in each layer by a simple linear transformation

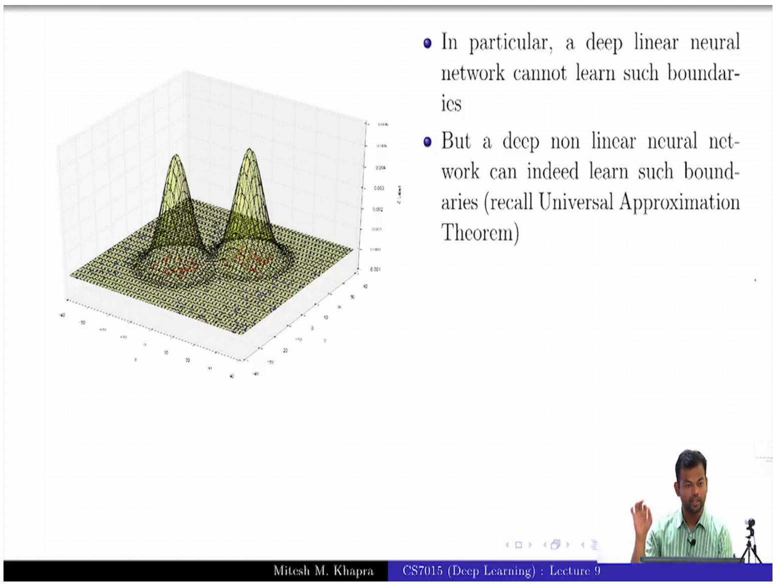
$$y = (w_4 * (w_3 * (w_2 * (w_1 x))))$$

- Then we will just learn y as a linear transformation of x
- In other words we will be constrained to learning linear decision boundaries
- We cannot learn arbitrary decision boundaries

So, what you are learning eventually? We will just y as a linear function of x and initially at some point we started off with such linear functions right; w transpose x in the case of perceptron and m p neurons.

So, what does that lead to what kind of decision boundaries does that lead to linear decision boundary right. So, if you do not have these nonlinearities we cannot have these arbitrary decision boundaries will only be left with linear decision boundaries.

(Refer Slide Time: 02:20)

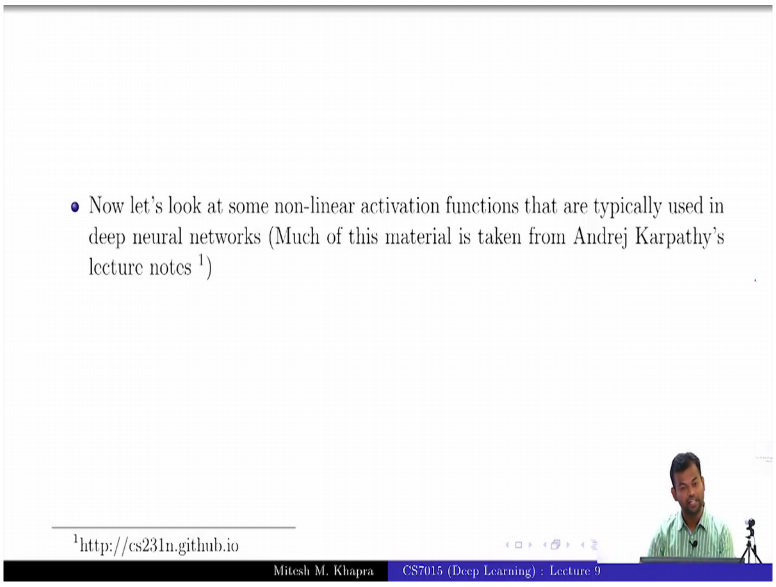


- In particular, a deep linear neural network cannot learn such boundaries
- But a deep non linear neural network can indeed learn such boundaries (recall Universal Approximation Theorem)

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

In particular, will not be able to solve this problem that we had right we were given some red and blue points and there was no way to draw a line such that the red points are separated from the blue points. What we needed is some kind of circles or ellipses to separate the red points from the blue points that cannot be done with linear decision boundaries. That can happen only, if you use a deep neural network with non-linear decision boundaries and we actually have a proof for that. What that proof the universal approximation theorem actually towards right.

(Refer Slide Time: 02:51)



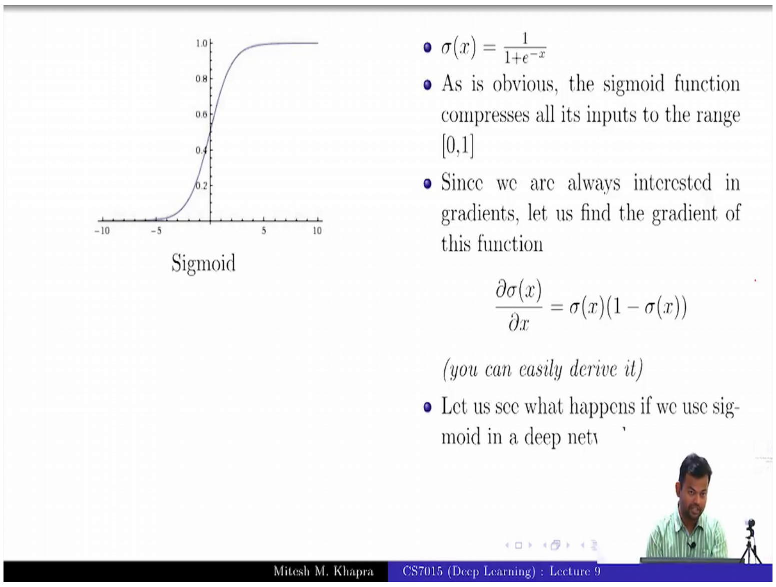
- Now let's look at some non-linear activation functions that are typically used in deep neural networks (Much of this material is taken from Andrej Karpathy's lecture notes [1])

[1]http://cs231n.github.io

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, that is why nonlinearities or the activation functions clear a very important role in the success of deep neural networks right. Hence you want to examine them very closely and see, what are the newer kinds of nonlinearities that have been proposed. So, we always start with the basics. So, will start with sigmoid see what are the problems with sigmoid? And then see what we can do to solve some of these problems? Ok.

(Refer Slide Time: 03:13)



- $\sigma(x) = \frac{1}{1+e^{-x}}$
- As is obvious, the sigmoid function compresses all its inputs to the range [0,1]
- Since we are always interested in gradients, let us find the gradient of this function

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$

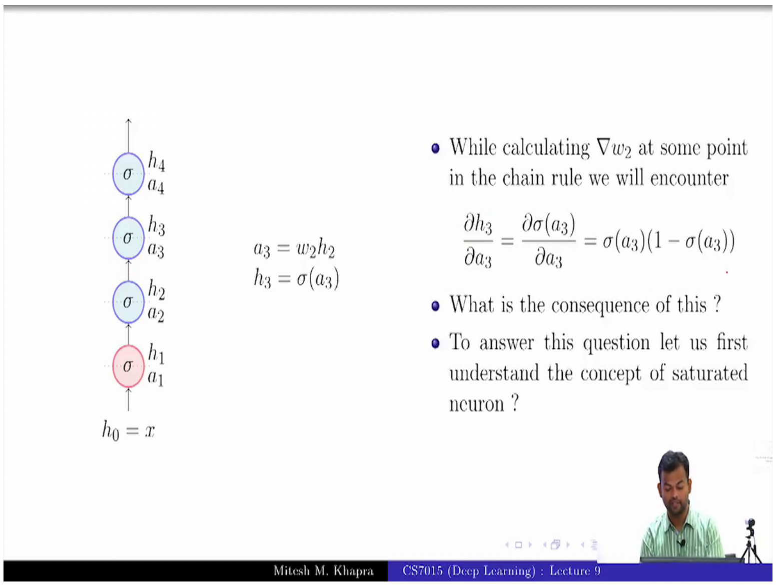
(you can easily derive it)

- Let us see what happens if we use sigmoid in a deep net

So, this is what the sigmoid function looks like you have seen it a million times and it actually constrains the input to 0 to 1 right. So, it takes some input and it constrains it 2 values between 0 to 1. Now since, we are always interested in gradients right. Because, the entire training and that is why I did that precursor in the first module the training always depends on gradients.

So, it is always important to look at what does the gradient look like? So, we know what the gradient looks like? We have computed this is just sigmoid of x into 1 minus sigmoid of x ok. So now let us see what happens if you use such a sigmoid neuron in a deep neural network ok.

(Refer Slide Time: 03:49)



$a_3 = w_2 h_2$
$h_3 = \sigma(a_3)$

- While calculating ∇w_2 at some point in the chain rule we will encounter

$$\frac{\partial h_3}{\partial a_3} = \frac{\partial \sigma(a_3)}{\partial a_3} = \sigma(a_3)(1 - \sigma(a_3))$$

- What is the consequence of this ?
- To answer this question let us first understand the concept of saturated neuron ?

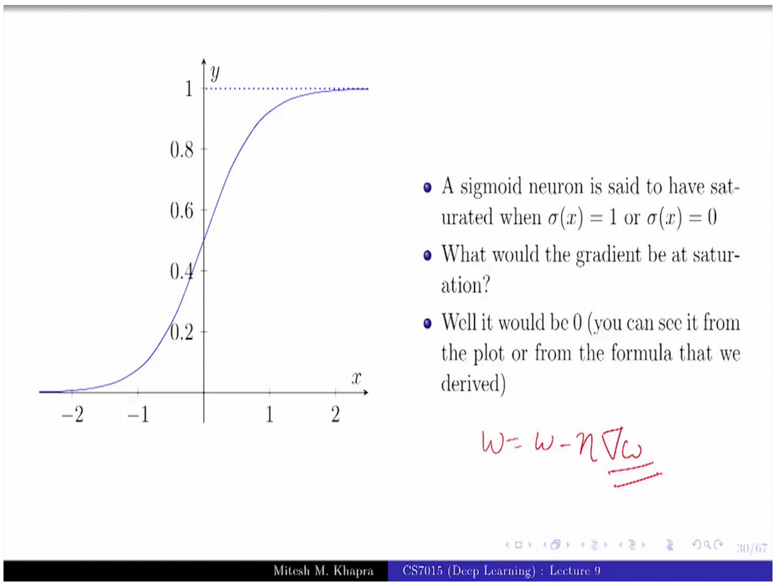
This is a deep neural network and without loss of generality I am going to use a thin deep network, but the same holds for a deep for a wide deep network also. So, suppose you are interested in computing the gradient with respect to w 2 right. At some point in the chain rule, you will have this term. How many of you are convinced about this? Ok, and that will lead to this could that cause a problem.

So, at some one of the terms in your chain rule is going to be this dou h 3 by dou a 3. I am assuming all of you are convinced about that and I have given you the exact formula for dou h 3 by dou a 3 will that lead to a problem.

Student: (Refer Time: 4:32).

Good. So, what is the consequence of this to answer this, we need to understand the concept of saturation right.

(Refer Slide Time: 04:40)



- A sigmoid neuron is said to have saturated when $\sigma(x) = 1$ or $\sigma(x) = 0$
- What would the gradient be at saturation?
- Well it would be 0 (you can see it from the plot or from the formula that we derived)

$w = w - \eta \nabla w$

So, a sigmoid neuron is said to have saturated. If it is output is 1 or 0 or rather close to 1 or close to 0 ok. What would happen in that case to the gradient?

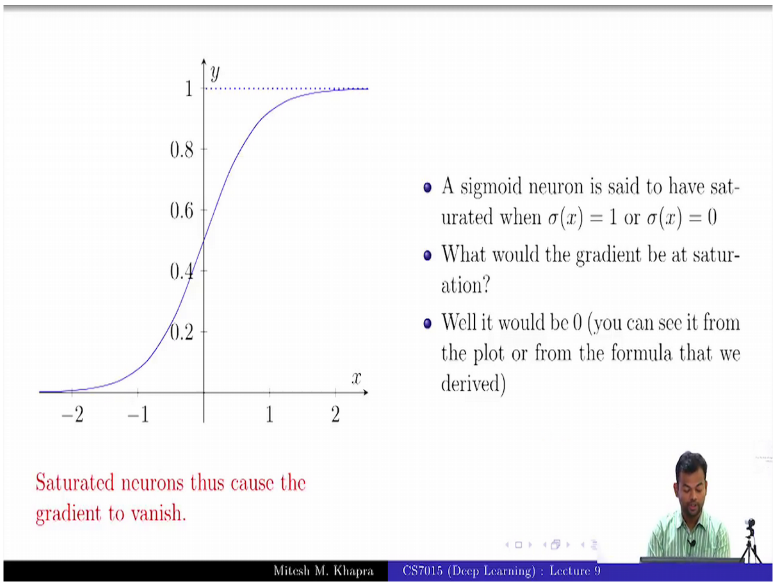
Student: (Refer Time: 4:48).

It will vanish right, because sigmoid of x into 1 minus sigmoid of x. So, it either extremes is going to vanish and you do not even need the formula for that. You can just see it from the diagram right. Because, the gradient here is going to be 0, that is obvious right. It just a what horizontal line.

So, this gradient would be 0. So fine, why does it bother us? What is our entire training premise based on gradients? Right, what does our update rule? What happens, if this guy is 0 no update e the weights just stay where they are right; that means, the training gets stalled right.

So think about this right, if all the neurons in your network have saturated; that means, all the weights the gradients will be 0; that means, all the weights will remain where they are you pass another input. Nothing is going to change right. It still be 0 so, if this neurons have saturated your training will just stalled. So, that was one of the reasons which is to cause problem in training deep neural networks earlier right.

(Refer Slide Time: 06:05)

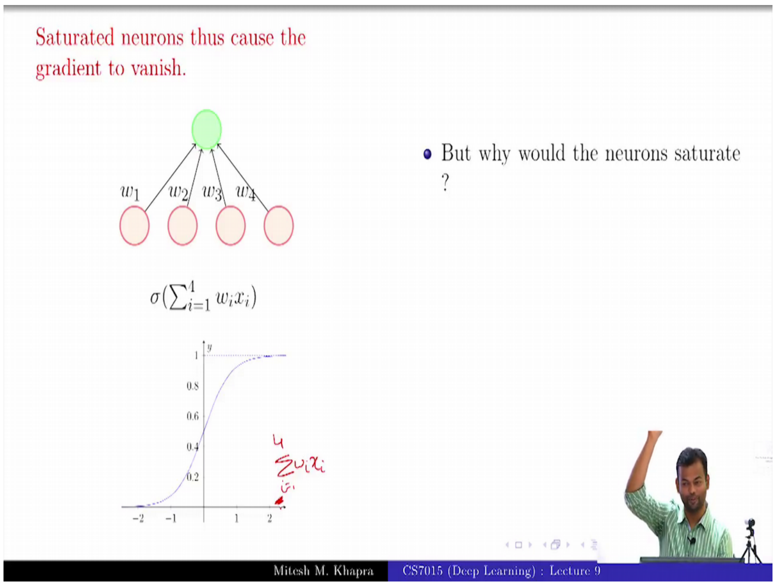

- A sigmoid neuron is said to have saturated when $\sigma(x) = 1$ or $\sigma(x) = 0$
- What would the gradient be at saturation?
- Well it would be 0 (you can see it from the plot or from the formula that we derived)

Saturated neurons thus cause the gradient to vanish.

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So that is one of the reason, why it was not converging? Because these weights used to these neurons is to saturate. So, this is one problem with sigmoid neurons a saturated sigmoid neuron can cause the gradient to vanish.

(Refer Slide Time: 06:15)


Saturated neurons thus cause the gradient to vanish.

$\sigma(\sum_{i=1}^{4} w_i x_i)$

- But why would the neurons saturate ?

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

But, why would the neurons saturated? I mean, what would cause them to saturate? Ok, this saturate find their gradients will vanish, but why would they saturate? We should be able to get some hints from the figure that has been drawn. So, this is actually that x needs to be changed.

So, on the x axis we have x quite obviously, but that has to be something else ah. So, what it is? What is happening is? What does the sigmoid neuron do? It takes this aggregate it or someone just disappear (Refer Time: 7:26). So, is it very boring today no right ok. So, you have this aggregated sum of the inputs. Once, you have that aggregation you applied the sigmoid now tell me when would it saturated.

Student: (Refer Time: 7:39).

When the aggregation is very large; that means, one of the 2 things could happen either the x's are very large or the w's are very large. Would the x is x is be large I see a lot of you saying no why?

Student: (Refer Time: 7:45).

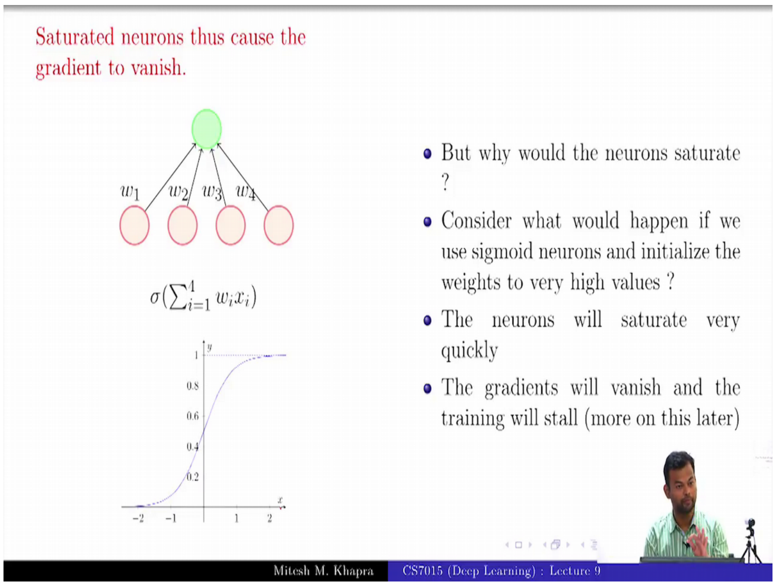
Good we normalize them right. We make sure they are between 0 to 1. So, we do not allow those arbitrary large values of pressure density and so on right. We make sure they are between 0 to 1. So, then the weights can be a problem right. Now, why would the weights be lies move later first?

Student: (Refer Time: 7:58).

If I initialize the weights to a large value, if I initialize all my weights in my infinite wisdom to a large value, what would happen right from the first training example itself. W i x i would take on a very large value and your neurons will start saturating right. So, imagine if all the weights throughout my network are initialized to large values.

Then right from training instance 0, my neurons will start saturating and I will not be able to train anything. How many of you experienced this while doing back propagation? And the others did not do the assignment, they copied it please raise your hands. How many of you experienced it? Now many more hands will be raised still now ok. Honest people that is a paradox, but.

(Refer Slide Time: 08:20)



Saturated neurons thus cause the gradient to vanish.

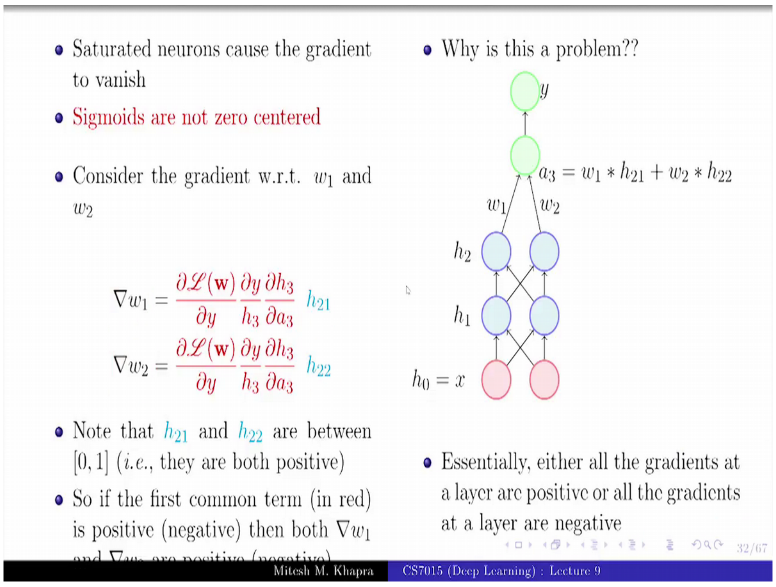
$\sigma(\sum_{i=1}^{4} w_i x_i)$

- But why would the neurons saturate ?
- Consider what would happen if we use sigmoid neurons and initialize the weights to very high values ?
- The neurons will saturate very quickly
- The gradients will vanish and the training will stall (more on this later)

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

Consider what would happen if you use sigmoid neurons and initialize the weights to a very high value. They will start saturating and hence, you will have this problem of vanishing gradients. Everyone gets this, so this is a problem at this sigmoid neurons.

(Refer Slide Time: 08:37)



- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered
- Consider the gradient w.r.t. w_1 and w_2

$$\nabla w_1 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} h_{21}$$
$$\nabla w_2 = \frac{\partial \mathscr{L}(\mathbf{w})}{\partial y} \frac{\partial y}{h_3} \frac{\partial h_3}{\partial a_3} h_{22}$$

- Note that h_{21} and h_{22} are between $[0,1]$ (*i.e.*, they are both positive)
- So if the first common term (in red) is positive (negative) then both ∇w_1 and ∇w_2 are positive (negative)
- Why is this a problem??
- Essentially, either all the gradients at a layer are positive or all the gradients at a layer are negative

32/67

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

The other problem with sigmoid neurons which is very interesting is that they are not 0 centered. What do I mean by that? They are not 0 center that is, what it ok. So, sigmoid is are not 0 centered. What do I mean with that? Mean by that they are not 0 centered; the value is between 0 to 1 right. So, the average cannot be 0 it is always going to be above 0 ok. Sigmoid neurons are always going to take on positive values between 0 to 1.

So, y is that a problem. So, that is an interesting explanation. Oh did I say that did I put the acknowledgements somewhere. So, all of this material that I have been talking about it is taken from Andrej Karpathys lecture notes. So, here is this interesting explanation for this.

So now, consider this particular network and I am going to focus only on this part; that means, the output layer and just the layer before that and the layer before that has these 2 weights w1 and w 2. I am going to focus on that.

So, to update these weights, I need to compute what I will carry a gun from next time? All this is on camera. So, what do we need to compute gradient? Ok now, you will answer. So, we need to compute the gradient with respect to w1 and w 2 and this is what it is going to look like? What is the red part and blue part? Why red and blue the red part is dash for both common for both right.

So, this is going to be common I do not know why I did that? Ok (Refer Time: 9:39) ok. So, this red part is common for both and what is the blue part actually? What is dou a 3 by dou w 1 h 2 1 and dou a 3 by dou W 2? Everyone gets this; you are reading this formula and telling me right.

So, dou a 3 by dou w 1 is just h 2 1 and dou a 3 by dou w 2 is just h 2 2 ok. So, let me just plug in those values and note that h 2 1 and h 3 are between 0 to 1. So, can you make some interesting commentary on this interesting, but useful not just philosophical stuff? That these 2 derivatives are for the weights at a given layer, I have just taken 2 weights, but I could have taken n weights and the same thing would have hold right.

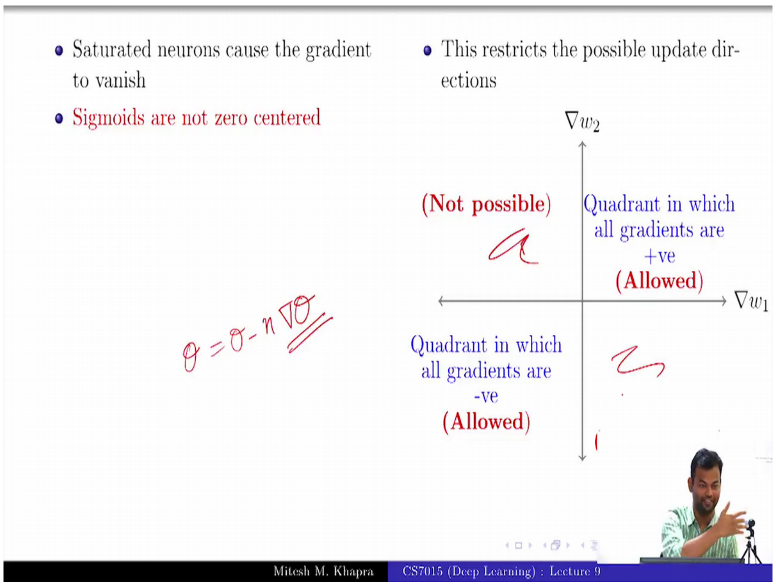
Because I know that the derivative is proportional to the input that it gets and the rest of the part is going to be constant because that is coming from the chain rule up to the previous layer right.

So now, what is happening because of that, just to make fun of you guys. I mean, if you get that sorry, good good yeah it is not very straightforward, but let us see. So, if the first common term in red is positive right. Then, what would happen to these 2 guys? They would both be positive right, because h 2 1 and h 2 2 are positive.

Now, the first common term in red is negative then, what would happen to these 2 guys? Both negative so; that means, the gradients of the weights at a particular layer where, either all be positive or they will all be negative you get that, that is because of this. Common part and the blue part the blue part we know is positive.

So, what matters is the common part and that common part can either be positive or negative for all of them together right; that means, for a given layer all the gradients at a layer are either positive or they are all negative. So, let us see what is the implication of that right?

(Refer Slide Time: 12:14)


- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered
- This restricts the possible update directions

∇w_2
∇w_1
(Not possible)
Quadrant in which all gradients are +ve (Allowed)
Quadrant in which all gradients are -ve (Allowed)
$\theta = \theta - \eta \nabla\theta$
Mitesh M. Khapra
CS7015 (Deep Learning) : Lecture 9

So, this actually restricts the possible update directions.

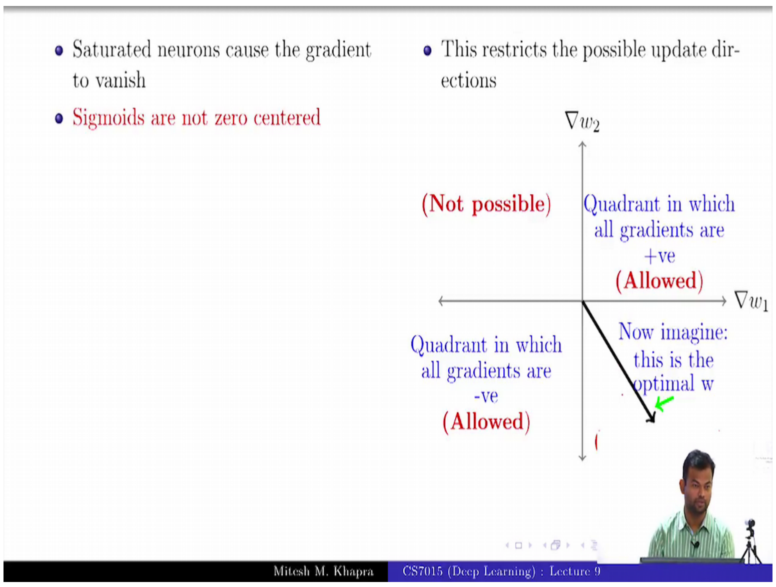
So, which is the quadrant? Which has all positive first? Ok sorry, for embarrassing yeah and all the negative is the third quadrant; that means, your movements can only happen in the first quadrant and the third quadrant. So, do you see a problem with this right? So, you are going to actually try to move that your theta.

Which is a collection of w 1 and w 2 is theta minus eta into the gradient right. And you know that, this vector which is the gradient vector can either be positive; that means can lie in the first quadrant or it can lie in the third quadrant. These movements are not possible; that means, there are certain turns or certain movements or certain directions, that I am not allowed to take. So, what would this mean it would take a dash time to converge.

Student: Longer time.

Longer time to converge right because, I am restricting my movement; so, imagine you have to go from destination to destination b and I say that you can never take a right turn right. And there is some going to be some problem, it will take longer to reach there unless the directions are to our left right unless your destination is (Refer Time: 13:22), but that will not happen ok.

(Refer Slide Time: 13:25)



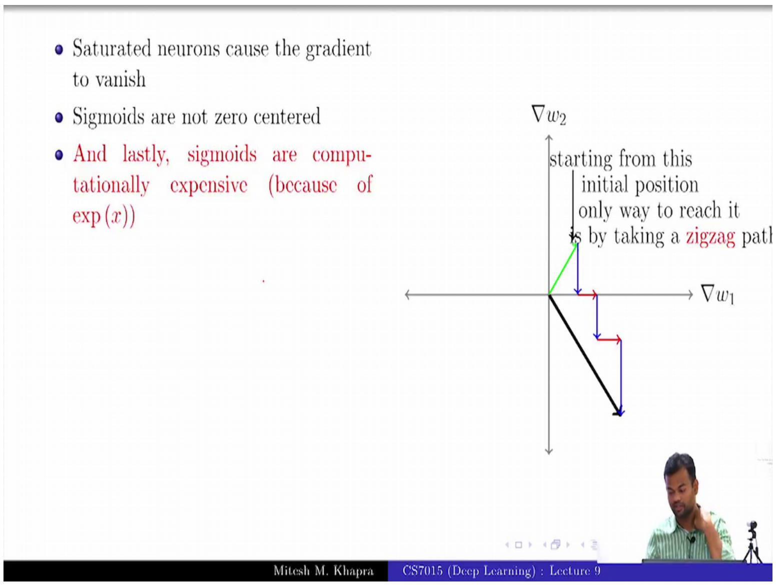
- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered

- This restricts the possible update directions

∇w_2
(Not possible)
Quadrant in which all gradients are +ve (Allowed)
∇w_1
Quadrant in which all gradients are -ve (Allowed)
Now imagine: this is the optimal w

So, suppose this is the optimal w star.

(Refer Slide Time: 13:29)



- Saturated neurons cause the gradient to vanish
- Sigmoids are not zero centered
- And lastly, sigmoids are computationally expensive (because of $\exp(x)$)

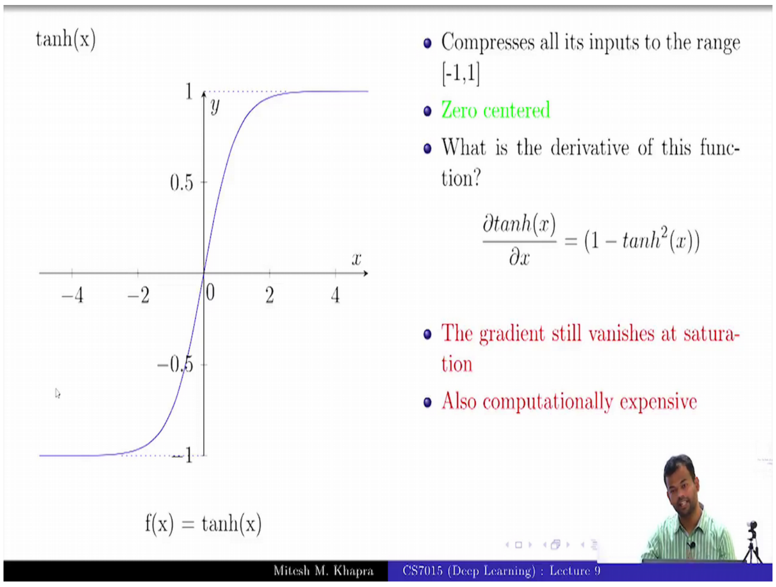
And we start with some random initialization because, that is why we are going to start. Then the only way I can reach it is, I may by making a series of this kind of movements right. As the exact pattern is, what will have to take? Because these are the only movements which are allowed or some movements which are allowed and it will lead to a certain cryptic pattern and I will not be able to have the complete freedom of moving in the direction. Which would have directly, taken me to the optimal. How many of you get this argument? Ok good.

So, that is a problem with something not being 0 center ok. And lastly sigmoids are expensive to compute because; you have to do this EXP right. It is not something as easier as something else that we will see in the lecture today ok. So, these are some problems with sigmoid functions. So now, peoples yeah.

Student: (Refer Time: 14:15).

So, we will ah. So, this is some issues that were they with sigmoid functions. So, this pointed that ok, maybe we should try better activation functions.

(Refer Slide Time: 14:23)



- Compresses all its inputs to the range [-1,1]
- Zero centered
- What is the derivative of this function?

$$\frac{\partial tanh(x)}{\partial x} = (1 - tanh^2(x))$$

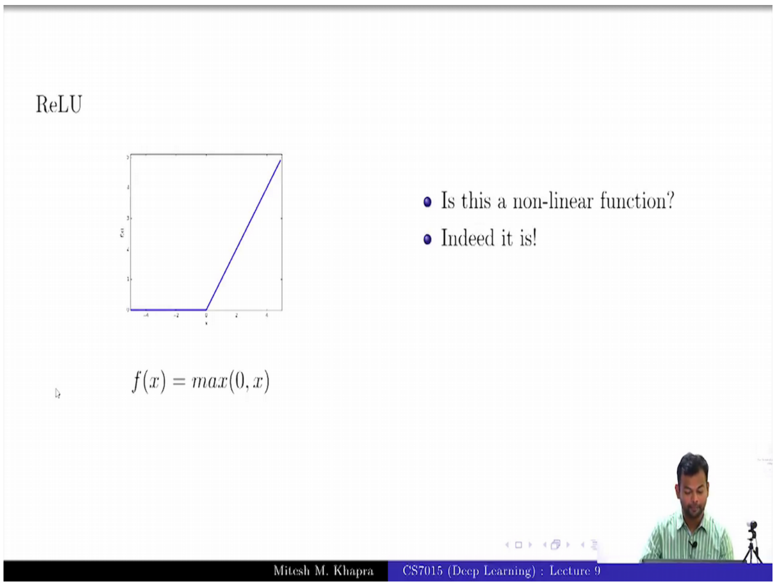
- The gradient still vanishes at saturation
- Also computationally expensive

That is why tanh become very popular, but tanh is not something which happened post 2006 right. So, this was like 92 or 93. When I think Yan Lacunae had started moving to tanh from sigmoid functions right. Now again here other inputs are compressed between minus point to 1 ok. Where inputs are now 0 centered which takes care of this problem which I mentioned at the end right.

That these directions of movements are constrained ok, and was the derivative of this function 1 minus tanh square right. What happens at saturation even without looking at the formula? The gradient would vanish to 0 right. So, the vanishing gradient problem is still there.

What you have solved is a problem of 0 centering? And that itself used to give better results than just using a sigmoid function, but it is still computationally expensive because, you still have to do these e raise 2 components right. The you still have to compute these exponential powers right. So, it is still computationally expensive.

(Refer Slide Time: 15:17)

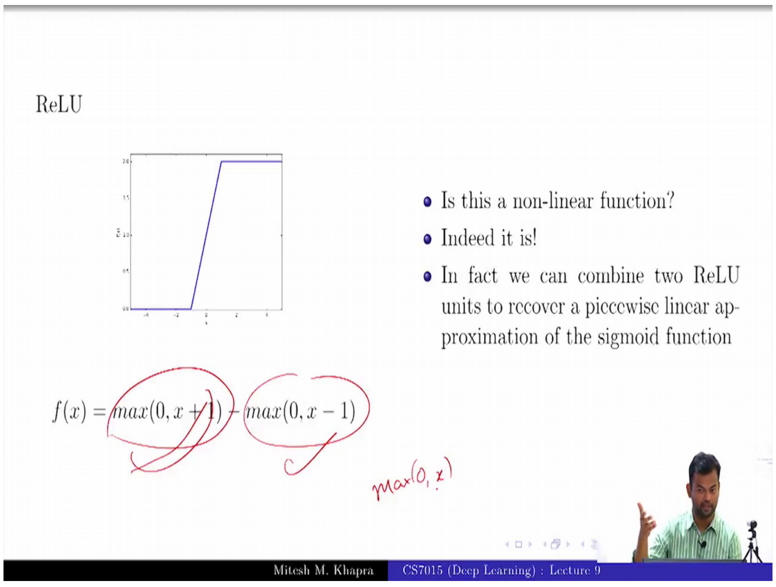

ReLU

- Is this a non-linear function?
- Indeed it is!

$f(x) = max(0, x)$

So, then in around 2012 I guess is when this ReLU was introduced in the context of convolutional neural networks right. And this is what the ReLU function actually looks like. Is this a non-linear function, it just looks like a line right? Why is it a non-linear function? It is a non-linear function right. Because, x is you cannot write x the output as a function of I mean as a linear transformation right. So, you have this 0. In fact, if you take 2 ReLU functions smartly.

(Refer Slide Time: 15:50)


ReLU

- Is this a non-linear function?
- Indeed it is!
- In fact we can combine two ReLU units to recover a piecewise linear approximation of the sigmoid function

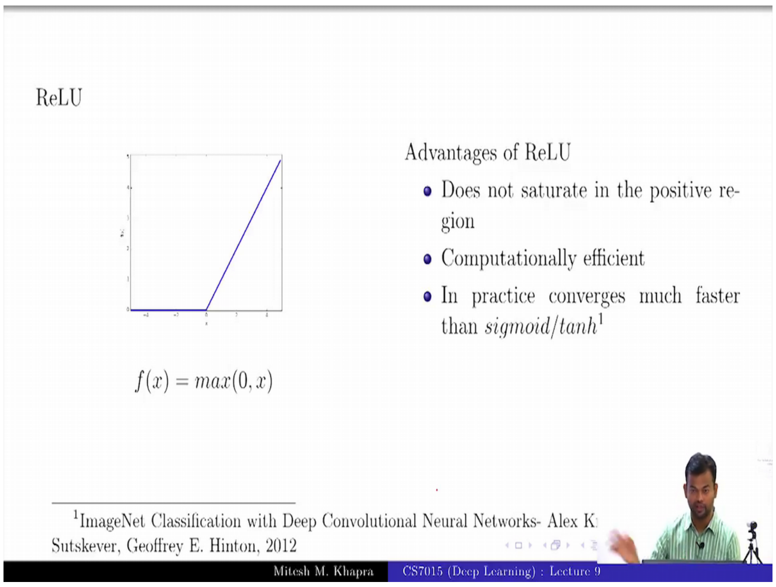
$f(x) = max(0, x + 1) - max(0, x - 1)$

$max(0, x)$

You can actually get the sigmoid I mean you can get an approximate for the sigmoid function. So, you can go back and check this right. So, if you take these 2 functions and subtract one from the other what is this? This is a ReLU function this is also a real function right.

So, I define ReLU as max of 0 comma x. So, both of these are ReLU functions some variant of that and now, if you subtract one from the other. You will actually get a approximation of the sigmoid function right. And this cannot happen if you have 2 linear functions take any 2 linear functions, you will not be able to get this kind of an approximation.

(Refer Slide Time: 16:27)



ReLU

$f(x) = max(0, x)$

Advantages of ReLU

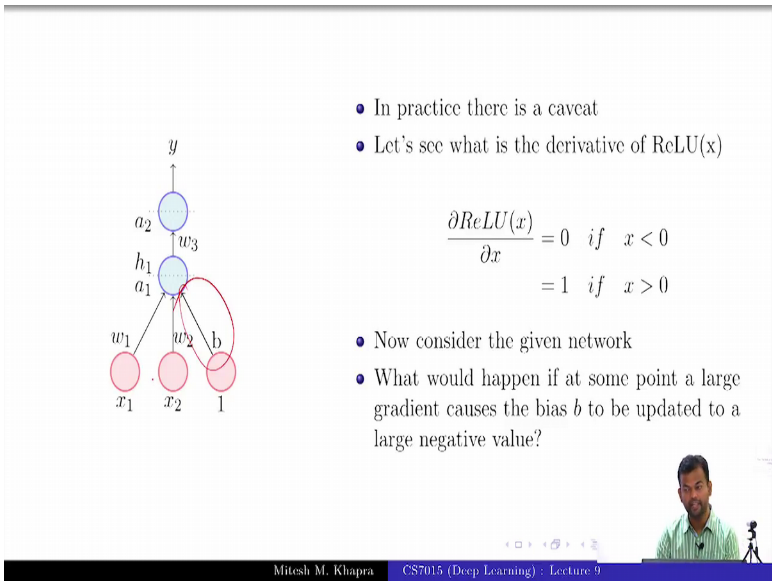
- Does not saturate in the positive region
- Computationally efficient
- In practice converges much faster than *sigmoid/tanh*[1]

[1]ImageNet Classification with Deep Convolutional Neural Networks- Alex K Sutskever, Geoffrey E. Hinton, 2012

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, ReLU is a non-linear function. What are the advantages of ReLU? One is, it does not saturate in the positive region right. It is computationally very efficient the output is either 0 or x. There is no powers nothing like that right, and it practice it converges much faster than sigmoid and tanh. So, that is what this 2012 paper show. And now, ReLU has actually become more or less the standard in all converge to neural networks, ok.

(Refer Slide Time: 16:53)


y
a_2
w_3
h_1
a_1
w_1
w_2
b
x_1
x_2
1
Mitesh M. Khapra
CS7015 (Deep Learning) : Lecture 9

- In practice there is a caveat
- Let's see what is the derivative of ReLU(x)

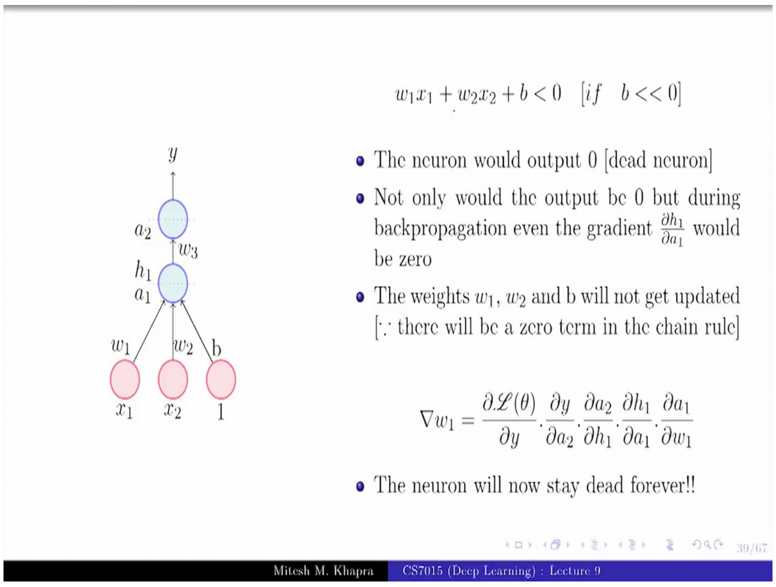
$$\frac{\partial ReLU(x)}{\partial x} = 0 \quad if \quad x < 0$$
$$= 1 \quad if \quad x > 0$$

- Now consider the given network
- What would happen if at some point a large gradient causes the bias b to be updated to a large negative value?

But there is still a caveat while using ReLU. So, the derivative of ReLU we can see that if x is less than 0. Then, the derivative is going to be 0 right and if x is greater than 0, then the derivative is going to be 1 and that straight away follows from the definition of ReLU which is 0 or x.

So, when it is 0 the derivative will be 0 and when it is x the derivative will be 1. So now, consider this given network and let us assume and this is not a very far faced assumed. Assumption it can happen in practice that, at some point a large gradient causes the bias b to be updated to a large negative value ok. So, what I am saying is that something happens and b gets updated to a large negative value.

(Refer Slide Time: 17:41)


y
a_2
w_3
h_1
a_1
w_1
w_2
b
x_1
x_2
1

$$w_1x_1 + w_2x_2 + b < 0 \quad [if \quad b << 0]$$

- The neuron would output 0 [dead neuron]
- Not only would the output be 0 but during backpropagation even the gradient $\frac{\partial h_1}{\partial a_1}$ would be zero
- The weights w_1, w_2 and b will not get updated [$\because$ there will be a zero term in the chain rule]

$$\nabla w_1 = \frac{\partial \mathscr{L}(\theta)}{\partial y} \cdot \frac{\partial y}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

- The neuron will now stay dead forever!!

Now, what would happen to this quantity remember this quantity which I have circled is actually the input to the blue colored ReLU neuron that I have. So, I am asking you what would happen to that input? That input would become negative.

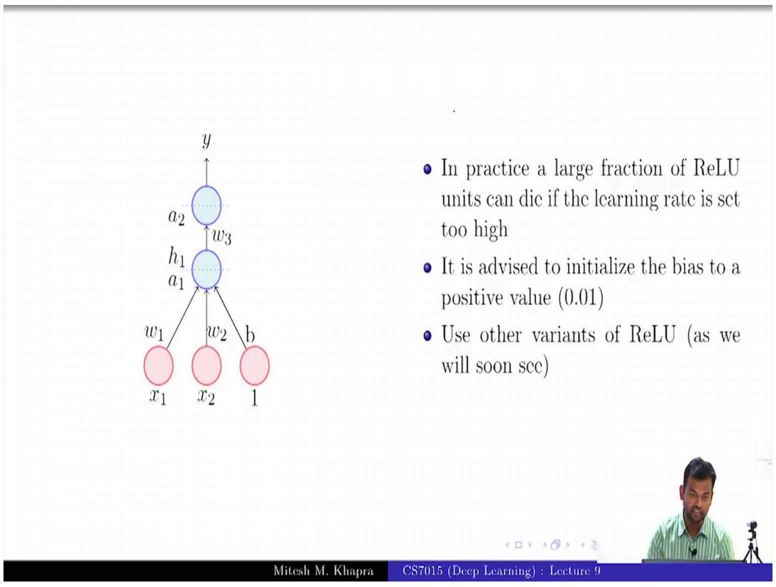
So, the neuron would output 0 and I am calling it a dead neuron. Why? If the input is 0, I mean is a input is negative then, the ReLU functions, output would be 0. What would happen to the gradients during back propagation 0? That means, what would happen to the weights?

Student: (Refer Time: 18:06).

Would not be updated right now, but that is fine right. If you give some other input this will recover. Why am I calling a dead means permanent? Right, unless you are in some fantasy world, but dead is dead right. So, why am I saying that it is dead? I could might as well I would give it a next input and then probably things would be ok, bias is still very negative because nothing is getting updated right or bias is still very negative. You know that x 1 and x 2 are constrained because, you have normalized them right and w 1 and w 2 have not been updated.

So, still the situation does not change. So, what happens is that once a ReLU neuron dies because, somewhere in the chain rule you got a 0. It will stay dead forever ok; it will never be able to come out of that. It will always produce a negative output; that means, that output will be clamped to 0; that means, no gradients will flow back and; that means, all the weights will not get updated connected that neuron right.

(Refer Slide Time: 19:16)



- In practice a large fraction of ReLU units can die if the learning rate is set too high
- It is advised to initialize the bias to a positive value (0.01)
- Use other variants of ReLU (as we will soon see)

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, in practice when you train a network with ReLU, you will observe that a large fraction of the units can die if the learning rate is set too high why this f condition?

Student: (Refer Time: 19:30).

What was the assumption that I made, that the bias receives a large negative update. And that is possible if your learning rate is very high. Because, you got some small negative gradient, but your learning rate blew it up.

Now, what is the practical implication of this? If a training a network and a large number of your ReLU neurons have died. What does it mean? Most parts of your network are dash useless; they are not learning any feature nothing right is all 0; that means, you have this large number of parameters versus getting wasted. Because, they feed into a ReLU you function and the ReLU function just keeps outputting 0.

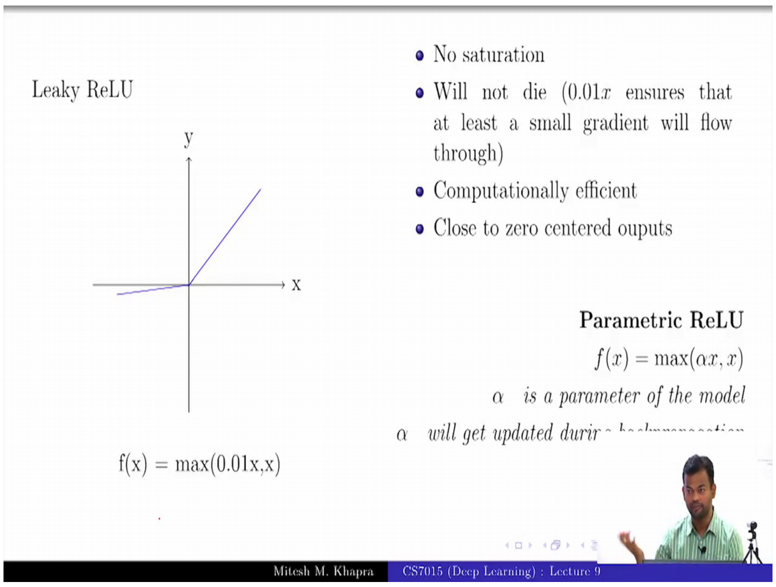
So, if you have n neurons in the particular layer and most of them are 0; that means, you are not really learning an n dimensional feature representation. You are just learning a much smaller feature representation right. So, can you give me a simple way of one simple way of avoiding this among many other ways?

Student: (Refer Time: 21:07).

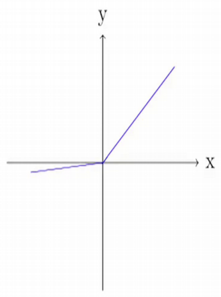
No dropout is statistical right, it is probabilistic this is like always dead one thing is to update the weight to a large to a positive value and 0.01 mind you is a large positive value right. Later on we will see y, but 0.01 is reasonably large ok. So, were going to initialize the bias to a positive value.

So, that even if this large negative gradient flows through there is still a chance that it will not become very negative. And hence, it will not mess up the things the way it does that is, one solution to that right, but still you will find that even after that the ReLU neuron a lot of those can die, but still in practice they work better for a deep convolutional neural network, and we can also use other variants of ReLU.

(Refer Slide Time: 21:31)



Leaky ReLU



$f(x) = \max(0.01x, x)$

- No saturation
- Will not die ($0.01x$ ensures that at least a small gradient will flow through)
- Computationally efficient
- Close to zero centered ouputs

Parametric ReLU

$f(x) = \max(\alpha x, x)$

α *is a parameter of the model*

α *will get updated durir*

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, there have been to avoid this dead neuron problem. There are other variants of ReLU, which have been proposed, and that is what we look at next. So, there is something known as a leaky ReLU is it obvious from the equation, what it does right. So, instead of producing 0 it will just produce a very small value proportional to the input. Now, what would happen to the gradients? They will not saturate right. Will have the gradient would be, if the input is negative what would the gradient be?

Student: (Refer Time: 21:56).

0.01 right so that means, some gradient will still flow through. How many if you get this? Right so; that means, if you use a leaky ReLU neuron some gradient would still flows through. So, just understand this trend right that ah. And this is I mean all this stuff is simple there is nothing great in this, but just put it in context right.

So, in 2006 to 2009 people realized. Ok now, we can trained networks and maybe whatever, we have done with unsupervised pre training actually corresponds to better initializations or better optimizations or better activations and so on.

So now let us try doing research in that. So, that led to the discovery of ReLU, now people started observing problems with ReLU and then proposed a variant of it which is leaky ReLU right. So, that is how this area has now become very prolific and grow right. So, we started off with this seed idea that, it is possible to train these deep neural networks and now we are trying to make arrive at better and better ways of doing it.

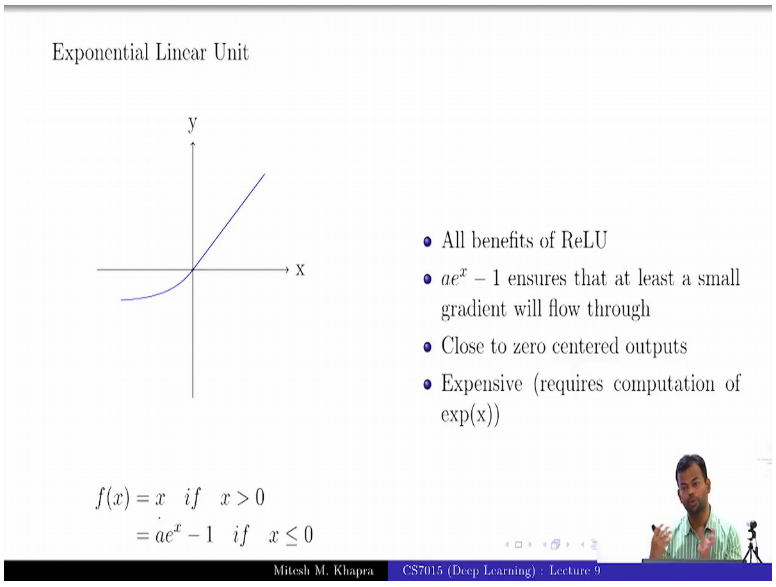
Making it more and more easier to train them and take care of some of these irregularities which existed earlier. So, one of them being sigmoid not being a very neat function to optimize with right. So, that is what all this is about individually all of these are probably easy for you to understand. Once you go back and look at the slides you all this is nothing great in this.

But, what I want you to really understand is this bigger picture of what is happening here as long as you get that time frame with. And of course, leaky ReLU is again computationally very efficient. There is no exponents no squares nothing like that, and it is close to 0 centered and it is still not 0 centered, but close to 0 centers because, you have outputs on both side and then someone came up with a generalization of this, which is parametric ReLU. So, y 0.01 make it alpha x and alpha will also be a.

Student: Parameter.

Parameter, it is a trainable parameter it is not a hyper parameter. Ok, how many of you know the difference between parameter and hyper parameter? Ok, you have used this in the back propagation as I am right. So, it is a trainable hyper parameter it will get optimized along with your other parameters in the network.

(Refer Slide Time: 23:55)



Exponential Linear Unit

- All benefits of ReLU
- $ae^x - 1$ ensures that at least a small gradient will flow through
- Close to zero centered outputs
- Expensive (requires computation of exp(x))

$$f(x) = x \quad if \quad x > 0$$
$$= ae^x - 1 \quad if \quad x \leq 0$$

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, then someone said leaky ReLU fine parametric ReLU is fine. Let us try to do exponential ReLU ok. So, it has all the benefits of ReLU it ensures that, at least a small gradient will flow through even when your inputs are negative; that means, it avoids this dead neuron problem again close to 0 centered outputs, but it is expensive because now we have added this exponential right.

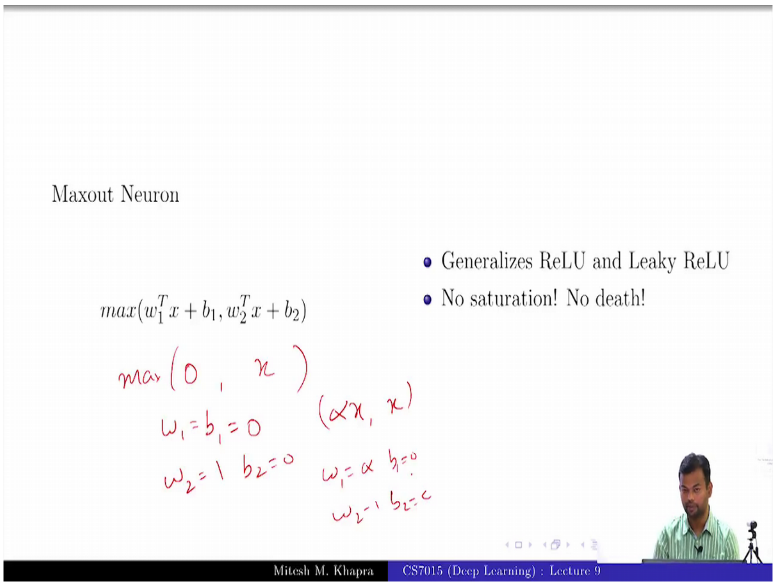
So, these are all ideas which came out during this period and all of them were shown to work better than the other and so on. And of course, at the end I have to tell you a final conclusion right. Whenever, I give you so many possibilities.

So, I have given you sigmoid tanh ReLU parametric leaky exponential. Now, what do you use? Right this the idea is not to confuse you, but to give you one solution, which would largely work yeah? What regularization?

Student: (Refer Time: 24:45).

Yeah, you could have done yeah that (Refer Time: 24:48) there is exactly. So, a lot of this research right, which has happened in this period. It is not a lot of it is juristic right; you solve one problem with ReLU. Ok, the neurons and saturated ok just make it something which does not saturated.

(Refer Slide Time: 25:05)



Maxout Neuron

$max(w_1^T x + b_1, w_2^T x + b_2)$

- Generalizes ReLU and Leaky ReLU
- No saturation! No death!

$max(0, x)$

$w_1 = b_1 = 0$

$w_2 = 1 \quad b_2 = 0$

$(\alpha x, x)$

$w_1 = \alpha \quad b_1 = 0$

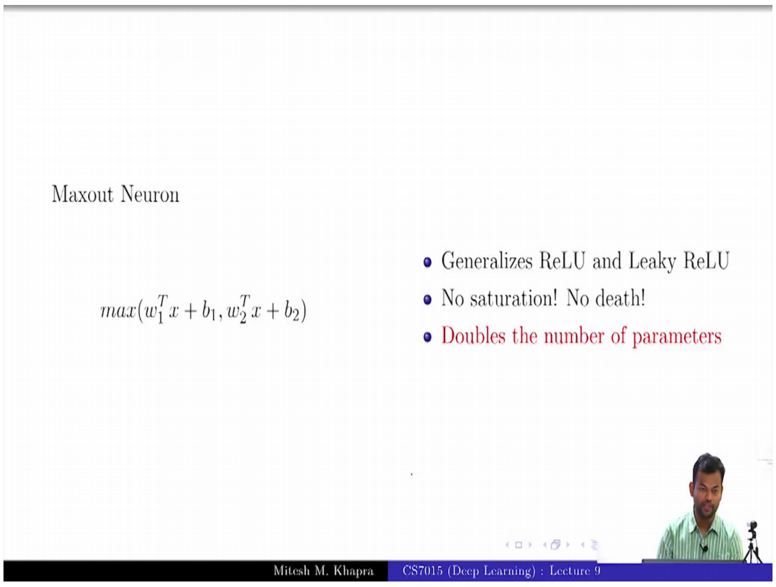
$w_2 = 1 \quad b_2 = 0$

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So that is there, it is possible that the other solutions would also go there is not that. This is the only solution which works now, then someone came out with max out neuron which is a generalization of ReLU and leaky ReLU. Why do I say it is a generalization? What was ReLU? That means, w 1 equal to b 1 equal to 0, w 2 equal to 1.

So, it is a special case of the max out neuron. What about leaky ReLU?, this was parametric value, but again what about. So now, what is happening w 1 equal to alpha b 1 equal to 0, W 2 equal to 1 b 2 equal to 0. So you see how it generalizes right. So, this is how these variants keep, kept coming up

(Refer Slide Time: 26:05)



Maxout Neuron

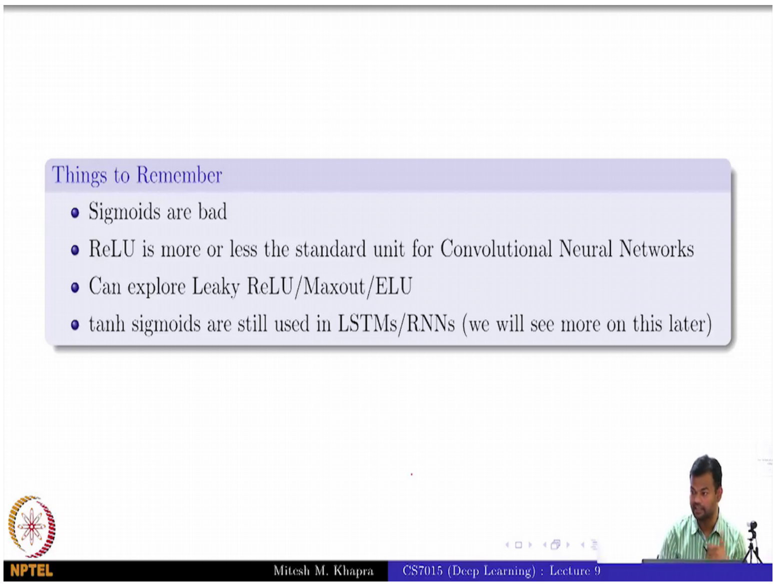
$max(w_1^T x + b_1, w_2^T x + b_2)$

- Generalizes ReLU and Leaky ReLU
- No saturation! No death!
- Doubles the number of parameters

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

Now, the problem of course is, doubles the number of parameters right because you earlier had only w transpose x plus b. Now, you have w 1 transpose b 1, w 2 transpose b 2 and so on right. So, it is actually doubling the number of parameters that you have.

(Refer Slide Time: 26:18)



Things to Remember

- Sigmoids are bad
- ReLU is more or less the standard unit for Convolutional Neural Networks
- Can explore Leaky ReLU/Maxout/ELU
- tanh sigmoids are still used in LSTMs/RNNs (we will see more on this later)

NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So now coming to the final conclusion of all of this right, what you need to remember is that sigmoids are bad.

So, no one uses sigmoids in convolutional neural networks. They still use somewhere, I am I am sorry, about this ReLU is more or less the standard unit for convolutional neural networks.

So, any standard CNN that you will pick up it will use ReLU as the activation function. If you want you can explore leaky ReLU max out ELU and so on, but it will require a lot of careful tuning. Say, if you want to use something out of the bulk box ReLU is just fine ReLU just works fine in practice despite all this dead neuron and other problems.

Student: (Refer Time: 27:20).

Yeah. So then, the argument for that is that, how often when you reach the point x equal to 0 right? So, the chance of that having is happening is very, very low. And if you get there, you can always approximate it by some epsilon or something and for that training instance just go on right. Any ways you are making so many approximations with stochastic and mini batch and so on.

So, this is one more approximation that is how people typically deal with it, but in most cases, it will not come in that point appearing is very low, but the question is valid and tanh sigmoids are still used in LSTM s and RNN s, which you will see at some later point in the course. So, there are a couple of more modules that I need to do. So, we just take a break here.