

**Artificial Intelligence**  
**Prof. Mausam**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology-Delhi**

**Lecture - 84**  
**Deep Learning: Perceptrons and Activation Functions**

So the topic of today is deep learning, right? I know that many of you have been waiting to hear about this particular topics. I am glad that we had enough time to study this together. And why are we here, right in terms of the narrative of the class. So the class has had a narrative right. And at some point, we realized that we have to do learning, right? We have realized this a multiple times.

But at least in the case of Minimax, when we were learning the utility function as well as in the case of reinforcement learning, when we realized that we have to generalize from the seen states to the unseen states. We recognized that we need to do learning right? Because the number of states in the world is going to be exorbitantly high. So high that, you know it might be more than the number of atoms in the universe.

But still humans can deal with that, right? If you if I take the scene in front of me, in a very high fidelity camera, the number of pixels will be of the order of you know whatever hundred thousands actually, right. And if you think about the RGB values for each particular pixel  $256 + 256 + 256$  right. That would be the number of possible states for one pixel raised to the power 100,000 for the total number of possible images with this kind of fidelity, right.

So even if we just look at images right, there will be just so many possible images and but humans are pretty good at understanding them. So humans do not have an issue with that. The human brain is extremely good at classifying images. The question that a lot of AI researchers right from the very beginning, so this is not a new development by the way. There have been at least three generations of neural networks.

Not only have there been three generations of AI, I mean this is what we have discussed at least, that earlier it was more logical, then it was more probabilistic. And now it is more neural. But actually, the neural networks themselves have been present

throughout these 60, 70 years. So the most interesting thing is that most of the algorithms that people do use today in neural networks are algorithms that were developed in the 80s and the 90s.

They were seminal algorithms which are ahead of their times at that time, right. Today, we have realized the importance right and we will talk more about that. But the intuition has been that okay, yeah is hard. Machine finds it hard to understand images, understand speech, understand text, whatever. But humans find it very easy.

**(Refer Slide Time: 03:07)**

The human brain is extremely good at classifying images

Can we develop classification methods by emulating the brain?

MPTEL

2

A small video inset in the bottom right corner shows a man in a light blue shirt standing at a podium, gesturing with his hands while speaking.

So what is it that is going on in our brain? And if we can learn something from there, can we put it in the machine in order to make a better AI system.

**(Refer Slide Time: 03:20)**

Brain Computer: What is it?

Human brain contains a massively interconnected net of  $10^{10}$ - $10^{11}$  (10 billion) neurons (cortical cells)

Biological Neuron  
The simple "arithmetic computing" element

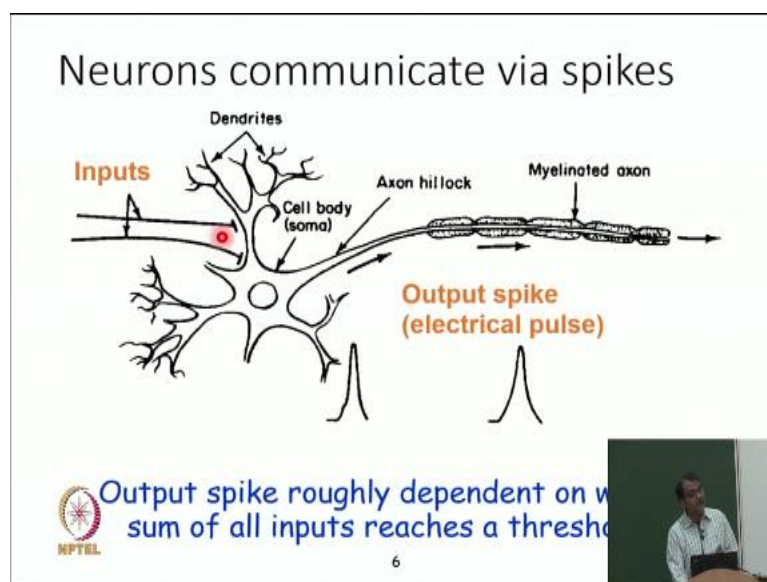
MPTEL

A small video inset in the bottom right corner shows the same man from the previous slide, now looking up and to the side while speaking.

And so the intuition was, let us see what the brain has, right. And a brain has a massively interconnected network of  $10$  to the power  $10$  to  $10$  to the power  $11$  neurons, right  $10$  billion neurons. But not only is it the number of neurons, but it is the amount of parallelism in these neurons that make the brain very different from a machine. In machine many things happen linearly. The amount of available memory is much limited.

You cannot do parallel computation because CPUs can only do right, so much parallelism. But human brain can, right. So there were a lot of attempts at saying, okay, let us say assume you know brain consists of neurons. These neurons are connected in this network. What is this neuron? And can we model it computationally? What, can we create a mathematical model for a neuron and then build it up from there.

**(Refer Slide Time: 04:22)**



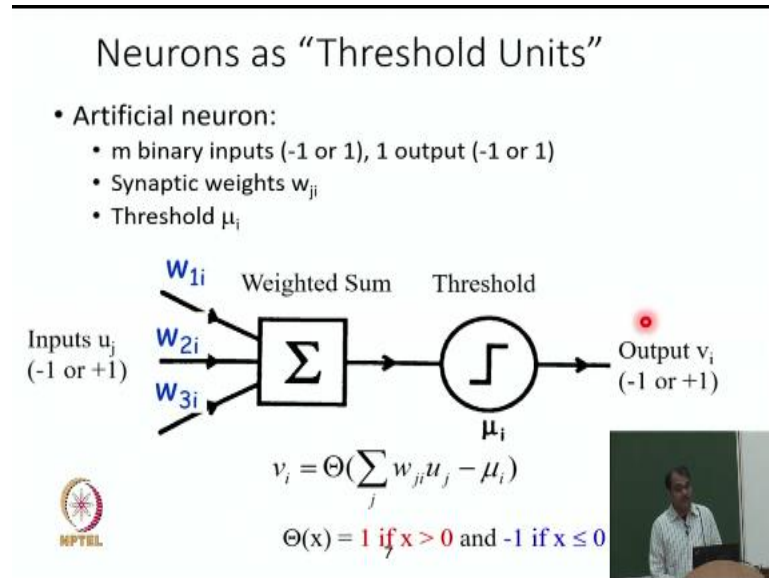
So the first intuition that you get is what is a neuron? Well, a neuron is something where lot of inputs come in from different neurons, or from sensors, right. And those inputs get processed into the nucleus of the neuron. And there from there, an electric pulse goes out to other neurons. This is sort of what a neural fundamental unit of a neural connection is in a brain, right.

And while all of these are grossed approximations, it was found that output spike, how much electrical signal do I send or whether I send a signal or not, is roughly dependent on the total strength of the inputs that were received by the neuron. So if a

very strong input comes in, then a strong output goes out. If a very weak input comes in, no output goes out. And so we said okay, I mean, this does not seem very hard.

So let us try to model it mathematically. Let us create an artificial neuron, okay.

(Refer Slide Time: 05:37)



And in this artificial neuron, let us say that it has some inputs. For example, let us say there are binary inputs -1 or 1 representing let us say true, false right. And then what we would do is we would give each input weight, some weight. We will not say that all inputs are equally important. Some inputs have been more important because they are coming from the eyes.

And some inputs may be less important because let us say they are coming from the nose or vice versa. So you will say that okay some inputs are more important some are less I will give weights to each input. And I will also say when should I send the output out. So I will create a threshold for that neuron.

And this can be represented in this kind of a schematic where I have a summation of all the inputs but not the original inputs but weighted sum of inputs. And then I pass it through a threshold that is it greater than  $\mu_i$  or not. And if it is, we will output +1 otherwise we will output -1. And my function if you think about it, a single neuron would be considered as a function from all the  $u_i$ 's or  $x_i$ 's whatever is your variable for inputs to an output single output, sending 1 or -1.

And it would be and  $\theta x$  is defined as it is 1 if  $x$  is greater than zero, and if it is -1, it is if it is less than equal to zero, then it is my -1, right. And if you think about it, this is actually a function. And it is a function, which, in your mind, you should be thinking, Oh, I want to create a function from my board state or my reinforcement learning grid world, or whatever it is to something, right.

And that something could be your Q value function, your policy function or whatnot. So eventually, I need to learn a function or everything can be casted as a function, right? Everything is a function. And now we cannot learn the function exactly because the number of inputs is humongously large, so we have to approximate. To approximate we have to define some structure over this function.

And we can learn the parameters like we did in Bayesian network, we define the structure using a Bayesian network and then we said what are the probabilities where we can learn it from data. So we will define the architecture or we will define a structure of my function approximator and then I will learn the parameters. For example, in this very simple artificial neuron, what are the parameters?

The weights and the threshold. These are the parameters for this very specific function approximate. Is this a linear function? Is this whole thing, which goes from the inputs to the outputs, would you call it a linear function? Is it a linear function, no. What would I what do I have to do in order to making to make it a linear function? There should not be any threshold. See this sum is a linear function.

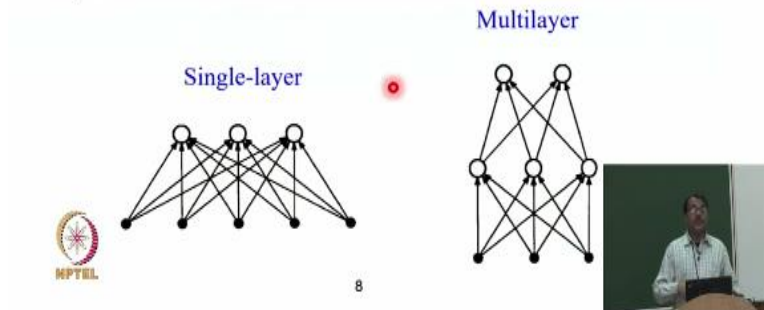
But as soon as we say okay if your sum is greater than  $\mu$  i output 1 as output -1, we have lost the linearity now there is a non-linearity. This unit of a neural network artificial neural network is called a perceptron, okay. That is the technical term for it. This is probably the first artificial neuron that was developed by AI researchers, okay. Of course, in our brain, we do not have a single neuron. We have a interconnection of neurons.

**(Refer Slide Time: 10:05)**

---

## “Perceptrons” for Classification

- Fancy name for a type of layered “feed-forward” networks (no loops)
- Uses artificial neurons (“units”) with binary inputs and outputs



So that is called neural networks, where you have many inputs, and then they go into these artificial neurons. And they have a connection between them. And finally, you output the final output that you want to output. And that is called a multilayer perceptron. You can also have a single layer perceptron where all the inputs go to one layer of neurons, and then you give the output.

So that is a single layer neural network. Now the first thing people said, okay, that sounds pretty good. And you know we can now start to emulate the brain. Let us try to understand what it can and cannot represent. Let us say I want to do classification. Let us say I want to, I have a set of inputs and I want to know whether they are in the class 1 or they are in the class -1.

So there are two classes and I want to find some separator in the input space between the two classes, right. What kind of functions can it represent and what kind of functions cannot can it not represent?

**(Refer Slide Time: 11:27)**

# Perceptrons and Classification

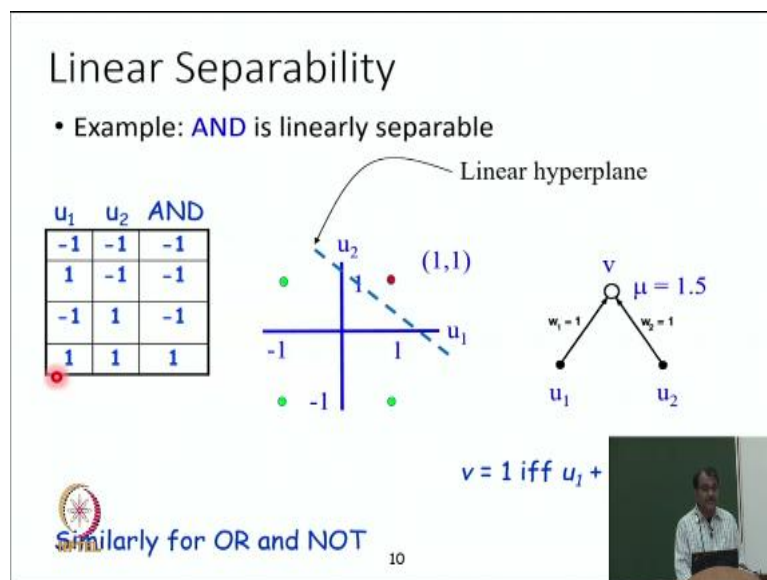
- Consider a single-layer perceptron
  - Weighted sum forms a *linear hyperplane*

$$\sum_j w_{ji} u_j - \mu_i = 0$$

- Everything *on one side* of this hyperplane is in *class 1* (output = +1) and everything *on other side* is *class 2* (output = -1)
- Any function that is linearly separable can be computed by a perceptron

So the first important observation that was made in this context is a function that is linearly separable can be computed by a perceptron.

(Refer Slide Time: 11:38)



For example, suppose I take the AND function. So you know these are my inputs. Now I will be in the -1 and 1 world rather than 0 and 1 world. So if it is -1, -1 I want to output -1. If it is 1, -1 I want to output -1. But if it is 1, 1 I want to output 1. So in other words, I want to output these three points as negative 1 class and this red point as positive 1 class. So if you look at it more carefully, there are many different ways to do this, right.

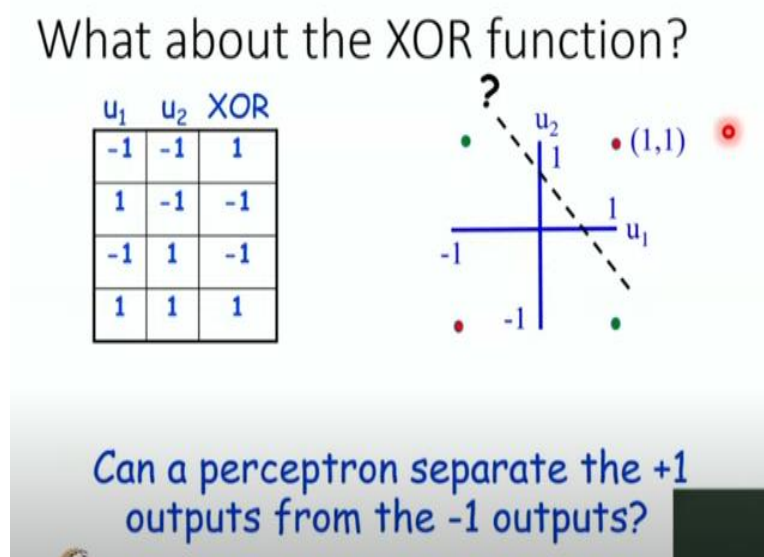
There may be one separator, which may be like this like a curved thing in the middle. There may be one separator, which may come from somewhere goes like this, and

then finally comes through. There can be many such separators. But what perceptron computes is a linear separator. For example, if you were if  $w_1$  is 1 and  $w_2$  is 1, and threshold is 1.5, then this creates this particular linear hyperplane.

Why, because if both of them are 1 then the sum is 2, and that is greater than 1.5. If both of them are -1, then this is -2. So +1.5 does not take it, sorry, it does it is not greater than 1.5. Even if one is 1 the other is -1 it is still zero. It is not greater than 1.5. Everybody with me? So this is good. We can compute the AND function using a perceptron. We can compute the ALL function using the perceptron.

What about the XOR function? This is a question. Yes. No, the question is will it always be some binary to you know 2 class problem? No we will generalize it, generalize the heck out of it. Any kind of input, any kind of output we will do a lot of things. In fact that is all the modern neural networks is all about, but we are building it slow, right. So can we compute the XOR function?

**(Refer Slide Time: 13:40)**



I want to compute the XOR function. What is the XOR function? They are 2 class problems. So both of them are same output the class 1, if both of them are different output the class -1 let us say the XNOT function. Sure. That is okay. You are saying convert -1 to 1 in  $u_2$ . Yeah, that can be done. It would not change anything XNOT becomes XOR let us say. It is the same thing, right?



So can we compute it using the perceptron? What do you think? Yes. We cannot do this with a perceptron she says. Why can we not do this with a perceptron? Because you are using a term which we have not defined, but that is the correct term. What is your name? Shaurya says, because they are not linearly separable. What does that mean? It means there does not exist a linear surface align or a hyperplane in a higher dimensional space, which can separate the positive class from the negative class.

You can try it. Can you come up with any line if you make a line here, the red green is here red green is here. If we make a line here then red is here and both greens are here. You have to make sort of like a curved surface, if you want to separate the reds from the green. Now that put cold water on research in neural networks.

**(Refer Slide Time: 15:35)**

## Linear Inseparability

- Perceptron with threshold units fails if classification task is not linearly separable
  - Example: XOR
  - No single line can separate the “yes” (+1) outputs from the “no” (-1) outputs!

Minsky and Papert's book showing such negative results put a damper on neural networks research for over a decade!

This is a very famous theorem by Marvin Minsky in his book on perceptrons, where he showed Minsky and Papert showed that perceptron with a threshold unit fails, if classification task is not linearly separate. Feels very obvious in hindsight right, because we are always computing a linear function. And then we are saying, are you greater than a value? Or you can take the value into the linear function and say, are you greater than zero?

Well, if you look at it, this space above this black line is when this value is greater than zero. And this space below the black line is when this value is less than zero or you can check it here, for example, for this blue line. So a perceptron eventually is

saying, look, you there is a linear thing. And if you are greater than zero that is one region if you are less than zero that is another region. And that is it, right?

And so we can only deal with linearly separable functions, which is not interesting, because in the world, there are highly nonlinear functions, right? We are in Delhi today. It is the pollution time. If we wanted to predict what is the total amount of pollution or AQI value of PM 2.5 value that a particular place will have, depending upon its wind speed, depending upon the you know time of the day depending upon whatever, right?

Would it be a linear function? Or if in the best world, you know you gave a very complicated board position and you said, okay, should everything depend linearly to the final value? Well, does not, right. And a linear function would be a gross approximation. So people said no we want to do better.



We want to go beyond the linear functions and we want to come up with nonlinear functions in a representation that can handle the nonlinear function, right. So there are two methods to deal with that. The first method is the idea of a multilayer perceptron.

**(Refer Slide Time: 17:57)**

### Idea 1: Multilayer Perceptrons

- Removes limitations of single-layer networks
  - Can solve XOR
- Example: Two-layer perceptron that computes XOR

• Output is +1 if and only if  $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$



What is a multilayer perceptron? It says do not just take it through one perceptron and give me the output, take through layers of perceptrons. For example, just two perceptrons, one after the other can help us solve the XOR function. We can look at it. It says output  $x$  plus output  $y$  minus two times the output  $x$  plus  $y$  minus 1.5. If it is

greater than 0.5 output 1 is output -1. Now it will take you some time to figure out what is going on.

So let us look at the lower neuron. When does it send 1 out? It sends 1 out when both of these are 1. Otherwise it sends -1. You can check that right. This is the same AND function actually here. So it send -1, that means in one case, the contribution will be -2, in other cases the contribution would be 2. But in the case when both of these are 1 you will get  $x + y - 2$  which would be 0. So it will not be greater than 0.5.

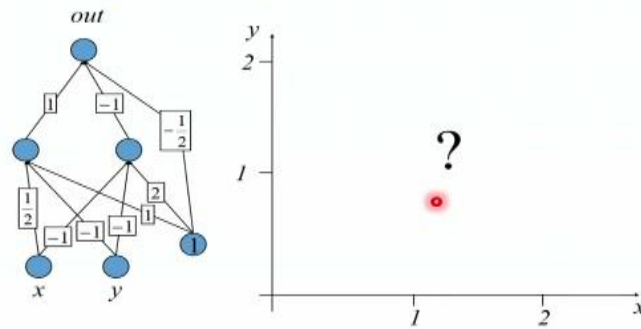
In all other cases, this term itself would be 2 and the other terms would be either 0 in case they are different, if they are different it will be 0. So that will be greater than 0.5. So you will send 1 out. In cases where they are both -1 then again they will be cancelling out because it will be  $2 - 2$  or sorry  $-2, -2$  which will be  $-4$ . Again it will not be greater than 0.5. So in the two cases where you have 0, 1 you get these two terms canceling out and this 2 taking over.

And in cases where in all other cases, you will have a 0 output or negative 1 output. You can double check that at home. This just two perceptrons can actually compute the XOR function. So while the Minsky and Papert theorem said we do not want to work on new networks, it cannot even represent a nonlinear function.

After a few years, people started coming back to neural networks because they realized oh, there is a very small fix. But then, if you have these layers of neurons, it was not easy because they had to then think about how do I learn the weights? Learning of weights becomes complicated, and we will talk about that in the next class. For now we are just understanding the expressiveness of the model, okay.

**(Refer Slide Time: 20:56)**

## Multilayer Perceptron: What does it do?



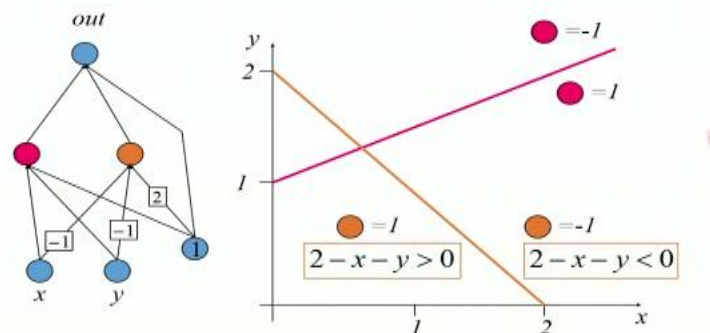
15



So what does a multilayer perceptron do? In order to understand this, let us look at this particular example. This is a very good example. Except now in for slides from now I will put the threshold inside as a single as a further input, right. You can always say that  $x + y$  greater than  $\mu$  can be written as  $x + y - \mu$  greater than zero. So that  $-\mu$  is what we will call the bias term. So we will just put the bias in the input okay.

(Refer Slide Time: 21:25)

## Multilayer Perceptron: What does it do?



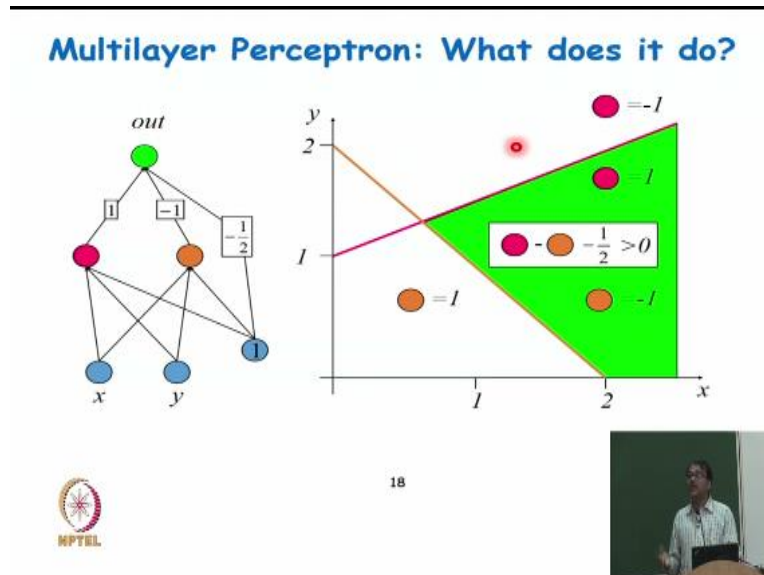
17



So this is a let us say a multilayer perceptron. Let us think about what is it doing. So the first neuron the pink neuron says half  $x$  minus  $y$  plus 1 should be is the input, right. Now it will have greater than zero or less than zero values and if you draw the line of that particular neuron, one side of it would be positive and another side of it would be negative. One side will be positive 1 the other side will be minus 1.

Similarly, if you look at the other neuron which is the orange neuron, one side of it would be greater than zero and one side of it would be less than zero. So here if you think about it, this is the part where orange is 1. This is the part where orange is -1. Similarly, this is the part where pink is -1. This is the part where pink is 1. Now let us check out the specific function pink minus orange minus half as the output function. Pink minus orange minus half.

(Refer Slide Time: 22:34)



Let me show you the answer. We can think about it together. This is the function that the green is computing. This is the final output. But where would it be 1 and where would it be -1? We need to think about that now. So where would it be 1? Let us think about that. For it to be 1, pink minus orange needs to be greater than half because these outputs 1 and -1. Pink has to be 1 and orange has to be -1.

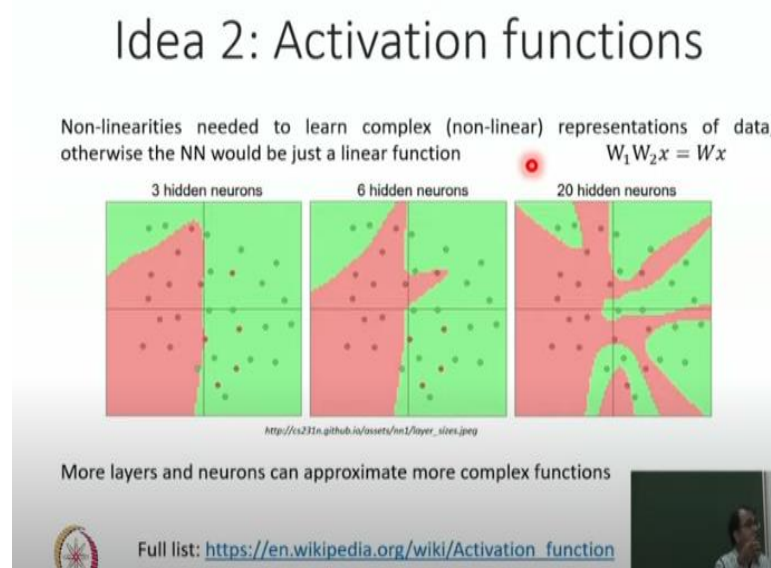
In all other cases that would not be the case. So that means that in at least in this example, the green will send 1 in this region where the two lines are creating a nonlinear region. So this is now a nonlinear function. Green is a function that separates this green region and this white region. So just by having two layers of neurons or perceptrons, we were able to compute this nonlinear function. In fact, and this is the interesting part.

There is a universal approximation theorem that says that given enough number of neurons, any two layer neural network can approximate a given function with arbitrary closeness. Neural networks is such a general representation that just by two

layers of neurons, I can represent or I can really well approximate any function whatsoever; linear nonlinear, highly complex in many different regions, whatever, just two layers.

That is the beauty and the power and the expressiveness of a neural network anyway. So this was our idea number one to deal with nonlinearity.

**(Refer Slide Time: 24:43)**



The other idea is a notion of activation function. So what are we saying so far, we are saying that your inputs go through a linear combination, and then we check it with a threshold and if the threshold is greater than something then we output 1 or we output -1. So we are sending a very discrete output out. We are sending 1 and -1, very discrete. But then people said there are many problems with sending the discrete output.

First of all, it cannot compute a continuous function like, you know a random function 3.7, right, it cannot do that. So there is a problem there. The other problem is that this perceptron is highly nonlinear, which basically means and non-differentiable. So which basically means the gradient here is zero, the gradient here is zero in the middle. At the point, the gradient is undefined.

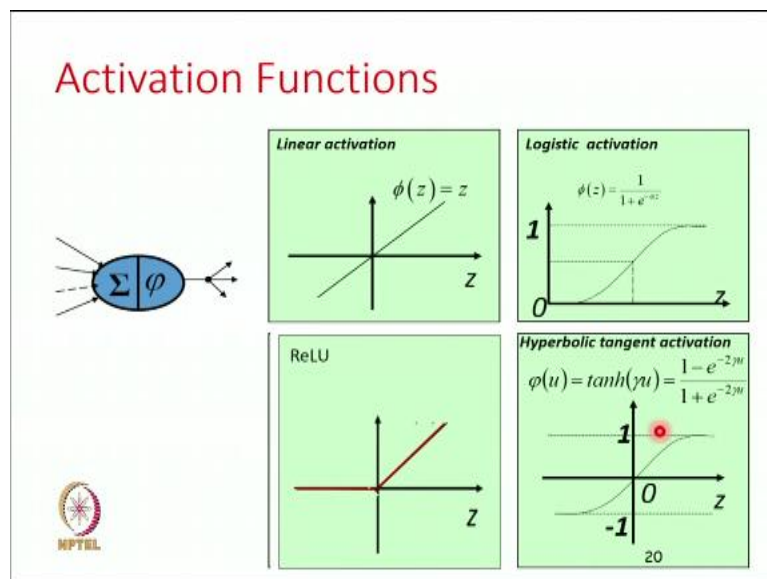
So like nothing is going on. I cannot figure out how I need to change my weights in order to get the function to do what it wants to do. So perceptron is very difficult to train. It is practically impossible to train, nobody uses perceptron. Well, okay, at least

you cannot use gradient based methods to train perceptrons. There are different methods to train perceptrons. They are good methods also.

But we will that you will study in a machine learning course. So I will backtrack, I would not say that nobody uses perceptrons but I will say that in the modern neural world perceptrons are not used very often. And yes, there exist training algorithms or perceptrons but we will not talk about that. We will say, well, we want the gradients to be slightly better defined.

So instead of a all 1 and sudden drop and all -1, let us make it smoother and continuous. And we do this by the notion of an activation function. So instead of saying either we send a signal out or we do not or we send -1 or +1 out, we are saying that we will send a continuous value out.

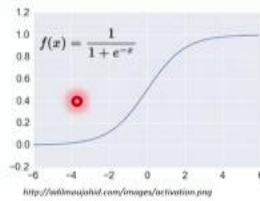
(Refer Slide Time: 26:47)



And the common activation functions are logistic activation, ReLU activation ReLU stands for rectified linear unit, and hyperbolic tangent activation function, short in short known as the tanh function. The tan hyperbolic tanh function, okay. I will very quickly talk about these. Again, this is not a full blown deep learning course. So I will very briefly give you intuitions about each of these activation functions.

(Refer Slide Time: 27:14)

## Activation: Sigmoid



Takes a real-valued number and "squashes" it into range between 0 and 1.

$$\mathbb{R}^n \rightarrow [0,1]$$

+ Nice interpretation as the **firing rate** of a neuron

- 0 = not firing at all
- 1 = fully firing

- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn when the neuron's activation are 0 or 1 (saturate)

- ☐ gradient at these regions almost zero
- ☐ almost no signal will flow to its weights
- ☐ if initial weights are too large then most neurons would saturate



So that when you talk to somebody you can come across as somebody who know things. But you do not know many things, just clarifying, okay. So this sigmoid function says,  $1$  by  $1$  plus  $e$  to the power  $-x$  output  $1$  by  $1$  plus  $e$  to the power  $-x$  where  $x$  is the cumulative weighted sum of the inputs. So if you carefully plot this curve, you find that this curve is always between zero and 1.

It starts out at minus infinity is zero you can check So if you put  $x$  equal to minus infinity, what do you get?  $1$  by  $1$  plus  $e$  to the power infinity which is infinity, so that is  $1$  by infinity which is zero. So at the limit, it starts out with zero and check what happens at  $x$  equal to infinity. You get  $1$  by  $1$  plus  $e$  to the power minus infinity.  $e$  to the power minus infinity is limited zero. So you get  $1$  by  $1$  plus zero you get 1.

So in the limit of minus infinity and infinity, you get zero and 1. Let us check out  $x$  equal to zero. It is a good point to check. So at  $x$  equal to zero what do we get?  $1$  by  $1$  plus  $e$  to the power zero.  $e$  to the power zero is 1. These are silly questions, but it just tells me that you are listening or not listening.  $e$  to the power zero is 1.  $1$  by  $1$  plus 1 is half. So what is it saying?

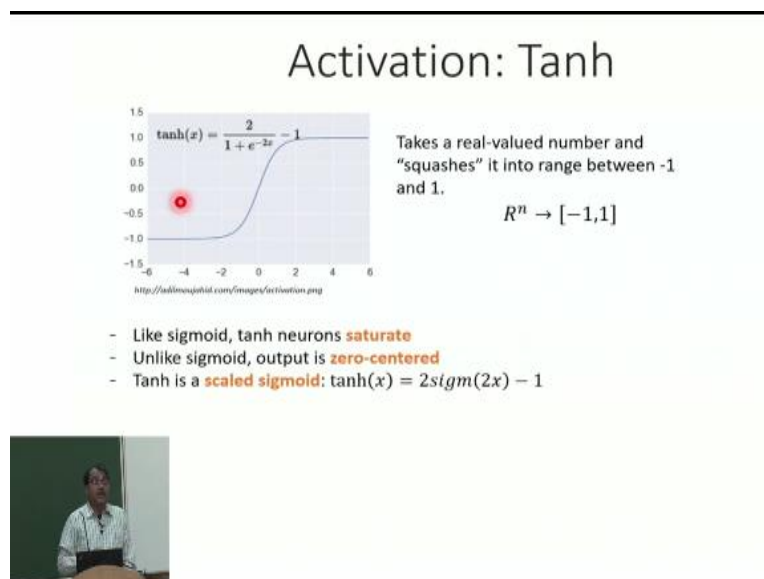
It is saying separate away inputs into two regions, when where it is less than half one where it is greater than half. It sort of actually to plot the curve it remains very close to zero for the longest time and very close to zero, around  $-4$ , it starts to go up. It goes up till around 4 and then it becomes very close to 1 and stays 1 throughout, okay.



And you can think of it as a neuron, which is not firing, not firing, not firing, not firing ambiguity region does not know maybe I should fire with small probability. Maybe I should fire with highest probability. Maybe I should fire with even higher probability. Now I am sure I should fire. Fire means send the signal out, okay. So if you think about gradients, the gradient very close to values 10, 20, 30 x equal to 10, 20, 30.

What is the gradient? Very close to zero. x less than -20 less than -10 what is the gradient? Very close to zero. When is the gradient interesting? Somewhere between -4 and 4, okay. So this is not an ideal algorithm, I mean ideal activation function because a lot of variants are zero but there will be some problem with every activation function. So we will keep moving.

**(Refer Slide Time: 30:03)**

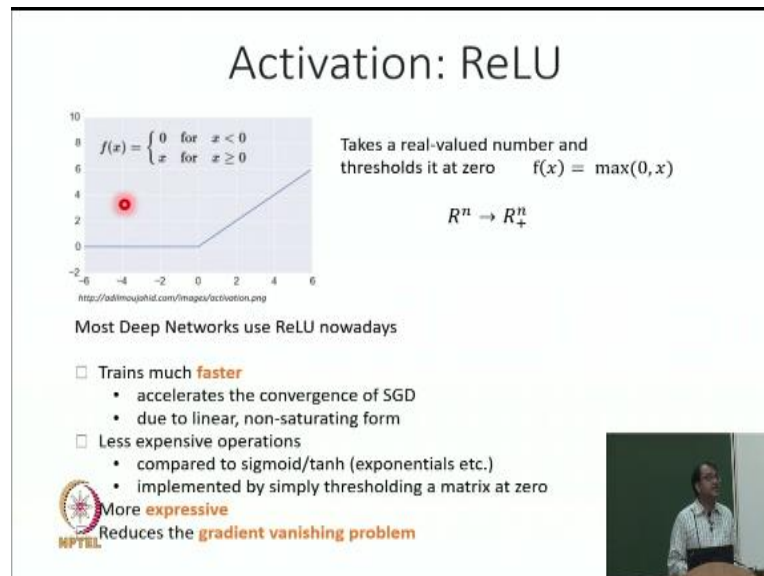


The other function is just take this zero to one curve and you know expand it vertically between -1 to 1. So how do you do that? You take the original curve, you multiply it by 2 and subtract 1 or something like that. So this is called the hyperbolic tan function. This is the smooth approximation of the perceptron function.

So perceptron was between -1 and 1 but a very quick sudden jump, a ladder jump, this is the smoother version of that. In my intuition, you use tanh function when you have to say, I have a positive influence on you, but I can also have a negative influence on you. But when you talk about sigmoid function, I use it when I only say I have a positive influence on you or no influence on you.

And my influence is also bounded in both these cases. But that intuitions you will generate over time. Then there is something called the rectified linear unit.

(Refer Slide Time: 31:10)



It is a linear function, but there is a non-differentiability at zero, where you say that from zero and below that, it is all I will not send any signal out. But in greater than zero point, I will send a linear signal out. So in the region zero to infinity, it is a linear function. In the region minus infinity to zero, it is a linear function. Such functions are called piecewise linear functions, but they are not. This is not a fully linear function, right?

It is a nonlinear function. The good news here is that your gradients in the positive region do not saturate. Your gradients do not become zero. Your gradient is 1 or -1. So that means that until you are in this positive region, you can keep training this weight much better, and we will talk about training later. So it was found that rectified linear units and networks of rectified linear units can train much better than networks of for example, sigmoid units. Okay.

By the way, this problem where your gradient goes close to zero. And your I mean your model does not train is called that gradient vanishing problem or the vanishing gradient problem. So vanishing gradient is a term. It is a technical term. If you are hearing it for the first time, I encourage you to go to Google or Wikipedia and read

about it. It is a very important phenomenon in making these deep neural networks train well, okay.