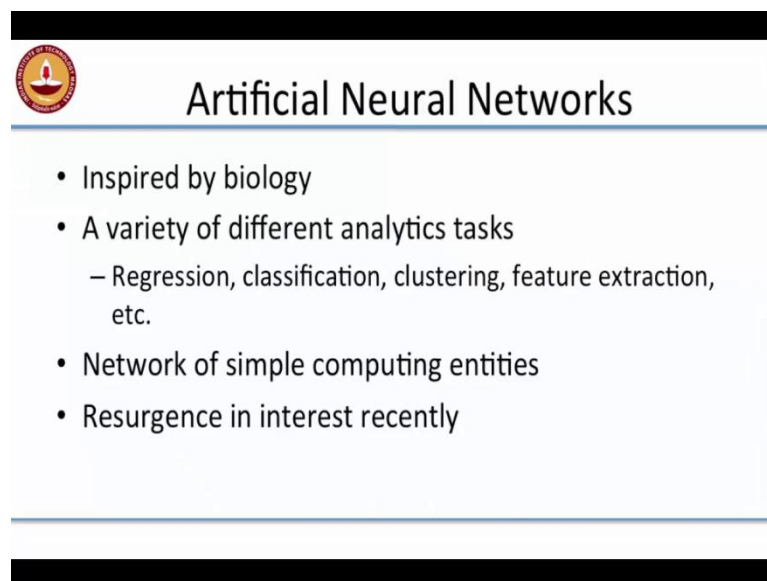


Introduction to Data Analytics
Prof. Nandan Sudarsanam and
Prof. B. Ravindran
Department of Management Studies and
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 06
Lecture – 34
Artificial Neural Networks

Hello and welcome to this module on Artificial Neural Networks.

(Refer Slide Time: 00:14)



The slide features a black header bar at the top. Below it, on the left, is the IIT Madras logo. To the right of the logo, the title 'Artificial Neural Networks' is displayed in a large, black, sans-serif font. A thin blue horizontal line separates the title from the content area. The content area has a light blue background and contains a bulleted list. The list items are: 'Inspired by biology', 'A variety of different analytics tasks' (with a sub-bullet 'Regression, classification, clustering, feature extraction, etc.'), 'Network of simple computing entities', and 'Resurgence in interest recently'. A thin blue horizontal line is at the bottom of the content area, and a black footer bar is at the very bottom.

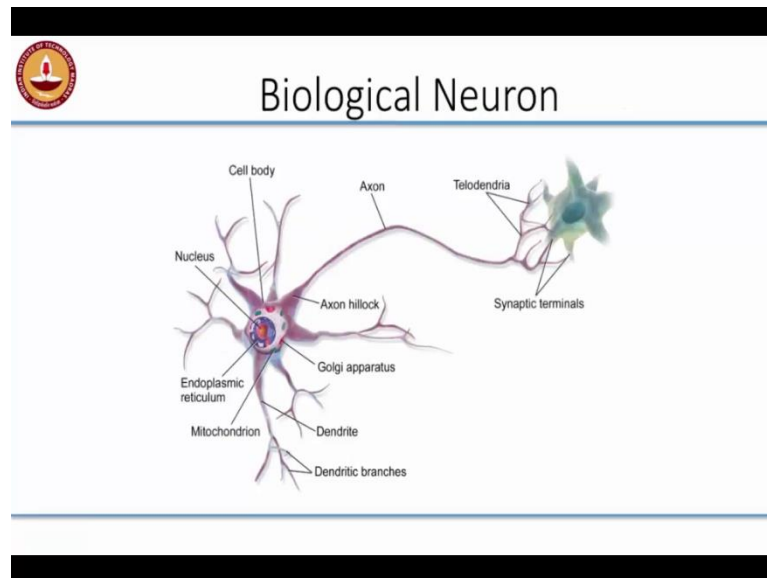
- Inspired by biology
- A variety of different analytics tasks
 - Regression, classification, clustering, feature extraction, etc.
- Network of simple computing entities
- Resurgence in interest recently

So, artificial neural networks, are computing models inspired by biology. So, we have neural network architectures that have been proposed for a variety of different analytics tasks like regression, classification, clustering, feature extraction, etc. So, these architectures essentially are networks of simple computing entities and so this is like a very simple threshold entities that are connected together in the specific network architecture that give rise to complex computing functionality.

Now, oscillate there has been significant resurgence in interest in artificial neural networks, especially under the domain of T networks about which we will see in one of the later modules. So, for this module and the discussion about artificial neural networks is concerned in this course, we will look at only the classification task and many of the ideas we talk about here for classification are generalizable to regression, like for while for the other kinds of analytics task we need different architectures and, but we are not

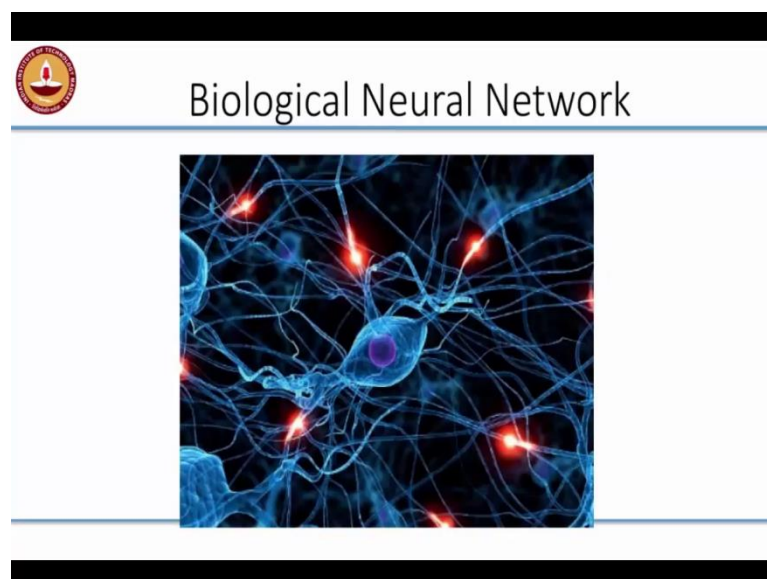
going to cover that in this course.

(Refer Slide Time: 01:22)



So, the inspiration comes from biological neuron. So, let us not worry about the complete complex structure of a neuron, what we really have to focus here is on the input and the output. So, the neuron receive inputs from the dendrites or from the dendrite branches from other neurons and when the input signals is above a certain threshold, it is going to produce an output, that is going to be transmitted via the synapses to neurons that are further down the line.

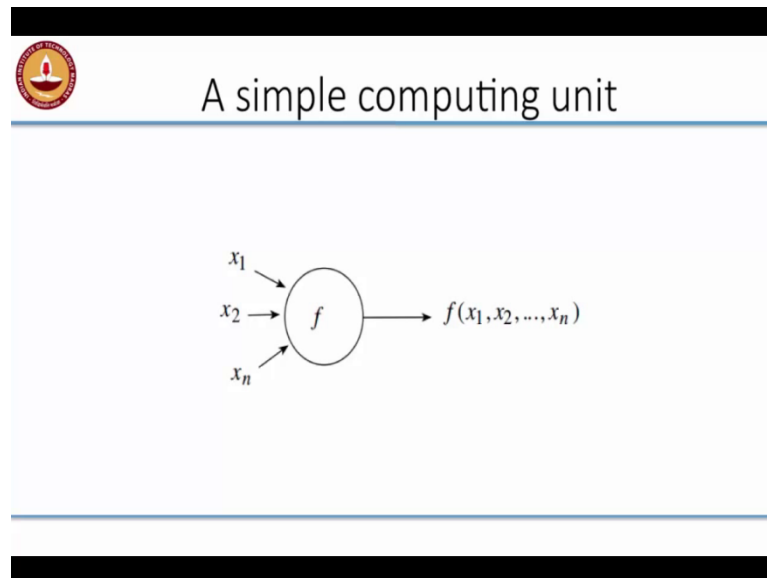
(Refer Slide Time: 01:54)



So, these connections to the dendrites and synapses are going to be result in a very

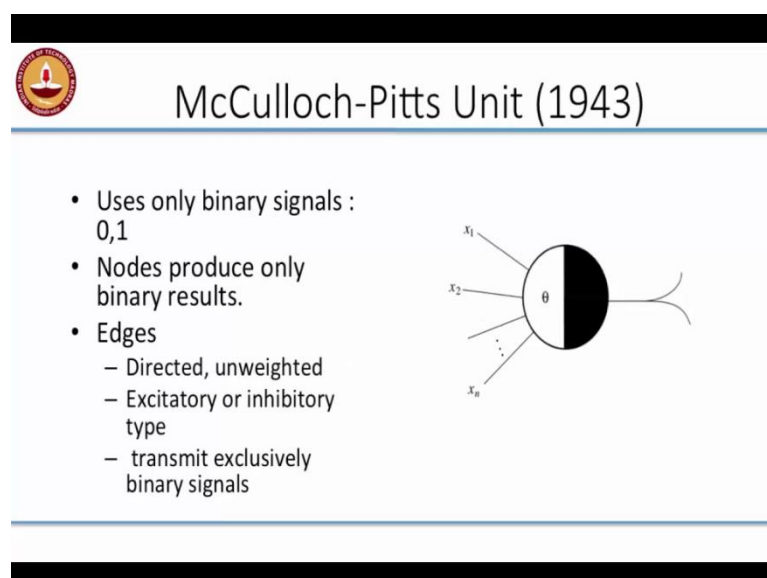
complex network and even though, the computing done by each element is very, very simple summation and thresholding. The sum total of this taken across the entire network can give rise to daily complex computations, which we will see.

(Refer Slide Time: 02:12)



So, the computing unit is something that is very simple. So, it is going to take a set of inputs x_1 to x_n and it is going to compute some functional form; its function is very simply incredible and then it will produce an output. So, we will look at what this function is going to be in detail in the next few slides.


(Refer Slide Time: 02:30)



So, the initial model for this for a biological neuron was proposed by McCulloch-Pitts in

1943 it is called the McCulloch-Pitts unit, it is only binary signals, so 0s and 1s. So, either an input is active, then these cases are represented by 1 or if it is not active, in this case it is represented by 0 and the nodes also produced only binary results. So, the outputs could be either 0s or 1. So, the edges between these different nodes were directed, unweighted, they could be of two types that could be excitatory or inhibitory and again I can mentioned earlier, the transfer binary signals.

(Refer Slide Time: 03:11)



Rule for evaluating the input

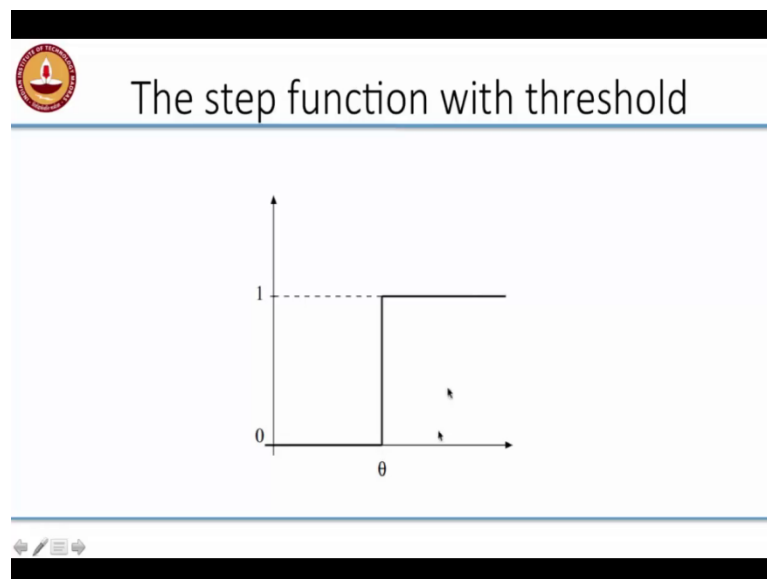
- Assume that a M-P unit gets inputs x_1, x_2, \dots, x_n through n excitatory edges and inputs y_1, y_2, \dots, y_m through m inhibitory edges.
- If $m \geq 1$ and at least one of the y_i is 1, the unit is inhibited and the result of the computation is 0.
- Otherwise, compute $x = x_1 + x_2 + \dots + x_n$. If $x \geq \theta$, the result is 1, else the result is 0.

So, what is the computation that happens here? So, I let assume that the McCulloch-Pitts unit gets inputs x_1 to x_n through n excitatory edges. So, these are positive edges and inputs y_1 to y_m through inhibitory edges. So; that means, these are edges that could produce the depression in the function or could actually stop the functioning of the neuron. So, the assumption that was made is, if m is greater than or equal to 1 that is at least one inhibitory edge and if any one of the inhibitory edges is 1, so if there is a one inhibitory input then the unit as a whole does not produce any output, regardless of what the inputs x_1 to x_n are.

If none of the inhibitory inputs are 1 or if there are no inhibitory inputs at all, the unit computes the summation of x_1 to x_n , let us call it x and if x is greater than the threshold that is specified for each unit, if it is greater than the threshold θ then the result of the computation is 1 as the result is 0. So, it is very simple, so essentially you can think of it as adding up all the inputs that come to the neuron and if the summation is greater than our threshold θ , your output will be 1; otherwise, your output will be 0. So, the inhibitory edges in some sense here acting act as a gating signal. So, if it is 1, the output is always

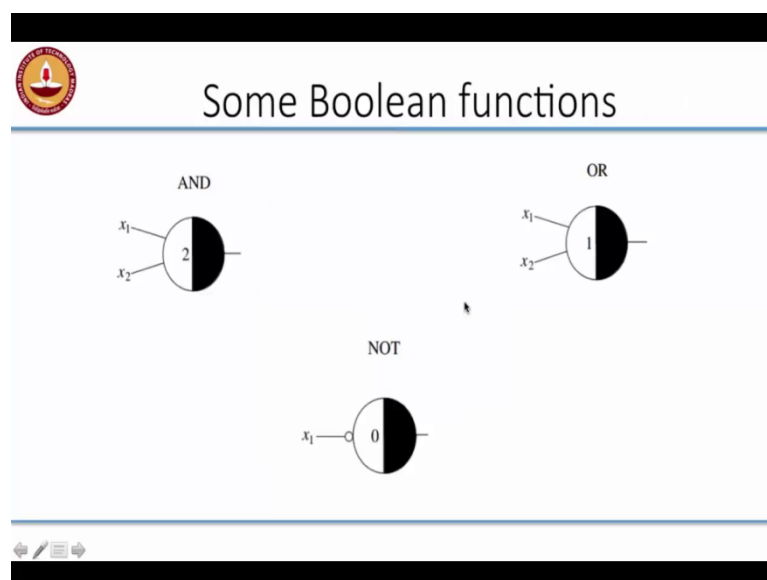
0, if the inhibitory is only 0 then the output is the result of the computation.

(Refer Slide Time: 04:43)



So, it is essentially the McCulloch-Pitts unit, it is implementing just threshold function. If the input is below θ you are going to see a output of 0, if the input is above θ you are going to see a output of 1, that this is essentially a step function.

(Refer Slide Time: 04:57)

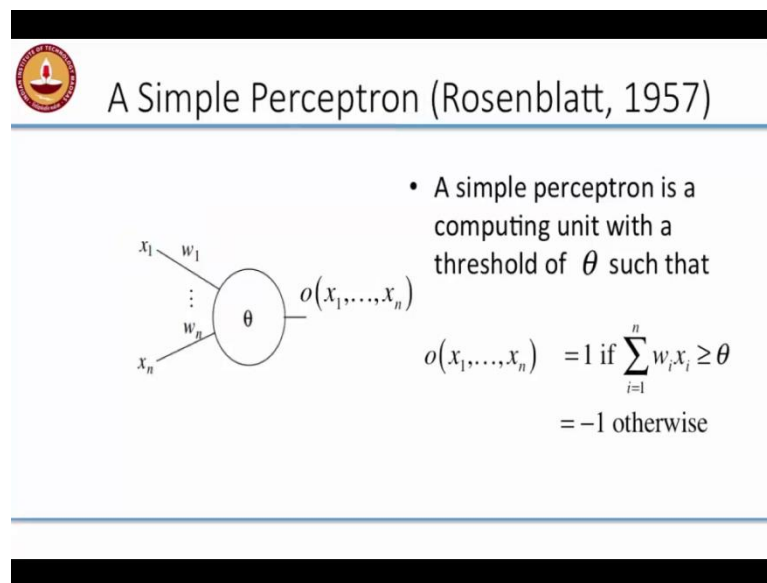


So, what kind of computations can you do with this? So, you can actually do almost all your familiar Boolean operations with the McCulloch-Pitts neuron. So, you can think of doing an AND operation, you have two inputs x_1 and x_2 and the threshold is set a 2. So, if only both x_1 and x_2 are one, so it will be greater than or equal to the threshold and

therefore, the output will be 1 and for implement in a OR you can set the threshold that one. So, if either x_1 or x_2 is 1 to the output will be 1 after complimenting a NOT unit it can implement the NOT unit by having x_1 act as an inhibitory input. So, this circle here indicates an inhibitory input.

So, if x_1 is 1; that means, they neuron and inhibitory output will be 0 on the other hand x_1 is 0 then the output will be whatever according to the result of the computation. But, we can see here that the threshold for this neuron set as 0 and that is for the output will be always 1 as long as there is no inhibitory input. So, if x_1 is 1 then the output will be 0, x_1 is 0 output will be 1, that how we have implemented NOT function. Now, once we unable to implement this kinds of AND, OR and NOT then you know that you can connect neurons together and then implement any Boolean function that we want and is this really we are interested in.

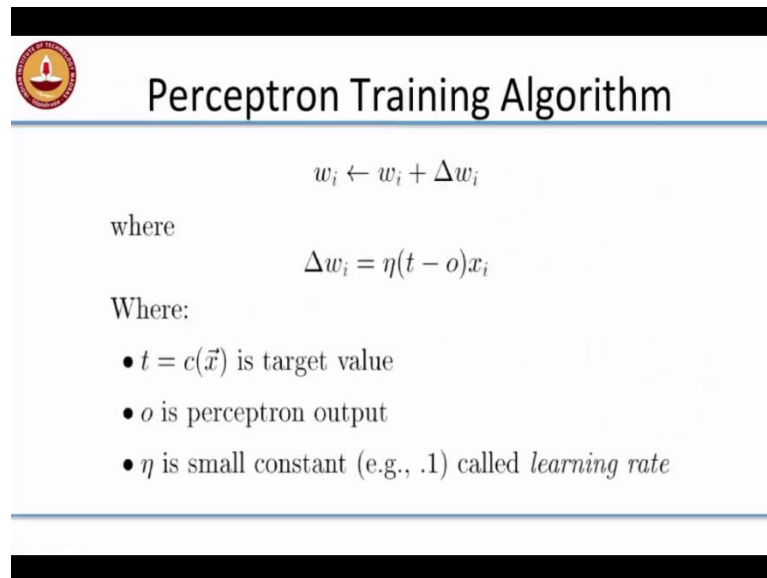
(Refer Slide Time: 06:27)




So, we are not really interested in that because we want to be able to do more complex classification problems, then we would like learn simple things like linear surfaces or more complex surfaces that is separate two classes. So, that is has been the goal of classification we have looked at so far. So, in 1957 rosenblatt proposed a very simple extension to the McCulloch-Pitts model which we called the perceptron, the more crucial thing what the perceptron is that a it introduced weights at the inputs, crucial differences from the perceptron from the McCulloch-Pitts module is that the perceptron introduced weights at the input.

And then the output could be either a one or a - one depending on whether the weighted sum of the inputs is greater than threshold that one that is the computing unit with a threshold θ . So, the output of the neuron is 1 if the weighted sum of the inputs is greater than or equal to θ is equal to - 1 otherwise.

(Refer Slide Time: 07:34)



The slide features a logo on the top left and a title "Perceptron Training Algorithm" in the top center. Below the title, the weight update rule is given as $w_i \leftarrow w_i + \Delta w_i$. This is followed by the word "where" and the formula $\Delta w_i = \eta(t - o)x_i$. Then, the word "Where:" is used to introduce a list of three items: $t = c(\vec{x})$ is target value, o is perceptron output, and η is small constant (e.g., .1) called *learning rate*.

 **Perceptron Training Algorithm**

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

So, what is the goal here in perceptron learning, when perceptron learning we are essentially trying to learn a hyper plane, trying to learn a separating surface as we have done in the past in the other classification problems, we are trying to learn the separating surface that can separate one class from the other. So, what would the classes be in our case, classes in our case would be + 1 and - 1. So, this essentially means if $w_i x_i$ is greater than equal to θ , it essentially defines the equation of a hyper plane as we have seen in the previous modules.

So, if this you can take the θ to the other side. So, we like $w_i x_i - \theta \geq 0$. So, we have seen that was greater than 0 to some one side of the hyper plane if it is lesser than 0 it is on other side of the hyper plane and we are going to say that data points to one side of the hyper plane belong to class 1 data points other side of the hyper plane belongs to class - 1. So, now, the question is given a set of training data that gives you the vector x and the decided output y .

How would we find these weights w_i 's such that the perceptron is actually implementing that hyper plane, implementing the right separating hyper plane. So, the weighted all this is follows, you start of the randomly initializing the weights to some value and then we

look at the prediction that is made by the way. So, the prediction that is made by the current setting of the weights, let us call it o and the target is the two class of the data point x . So, with this, it will be $+1$ or -1 and likewise o is also $+1$ or -1 . So, your goal is to make sure that here perceptron output matches the target value.

So, the perceptron training algorithm has a very simple rule. So, at every presentation of an input point, we change the weights by an amount that is proportional to difference between the target value and the actual output produce times that the input on the particular it. So, w_i changes by an amount that is proportional to $(t - o) * x_i$. So, η here is a small constant may be 0.1 or 0.01 as called the learning rate.

So, one thing to note here if for a particular input x_i will produce the correct output. So, the class is -1 and I produce -1 , the class is $+1$ and I produce $+1$. This expression evaluates to 0. You can see that this expression evaluates to 0 and therefore no changes in the weights will happen. So, essentially what happens here is you change the weights only whenever you make a mistake and that to you change the weight proportional to the input variable. So, if x_i is say a small value say 0.1 or 0.2 then will be changes in the weight will be small and as for as the poster when x_i is the large value let us say 1 or 0.95 and things like that then the change it be next will be large.

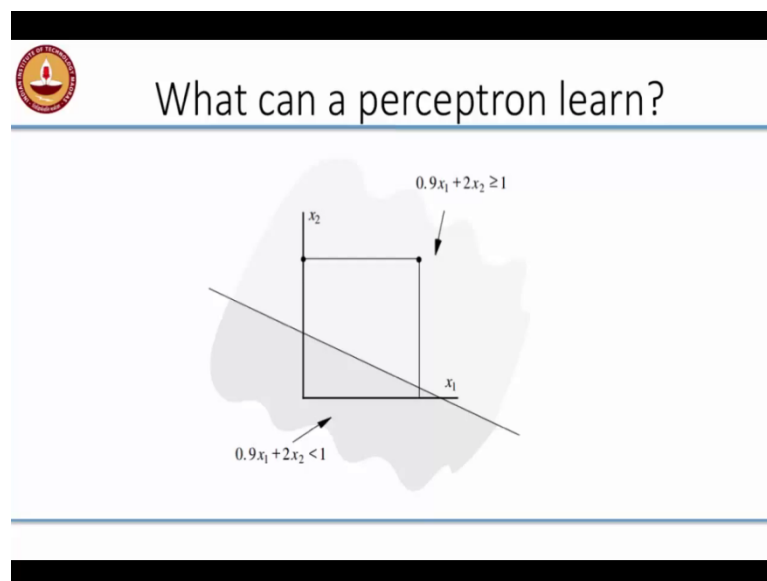
So, this essentially because the larger the input variable the more important it is going to be in the production of the output at least the way we are set up this perceptron. So, that is essentially the simple training rule. So, whenever you make a mistake, you take the vector for which we have made a mistake add some small fraction of that vector to the weights.

(Refer Slide Time: 11:14)



So, this looks like a very simple rule, but then back in 50's this perceptron's created a lot of human cried the people saw that the perceptron's by the able to learn from scratch trying to solve something which are considered hard learning problems and then they used the perceptron's they were able to solve that, so much so you can see here the hype was that they are going to build the computer that expects to be able to walk, talk, see, write, weight reduce itself and be conscious of it is existence, such the significant amount of hype and it is always hard to live up to any height that this proportionate and to the actual effect that was achieve that point.

(Refer Slide Time: 11:59)

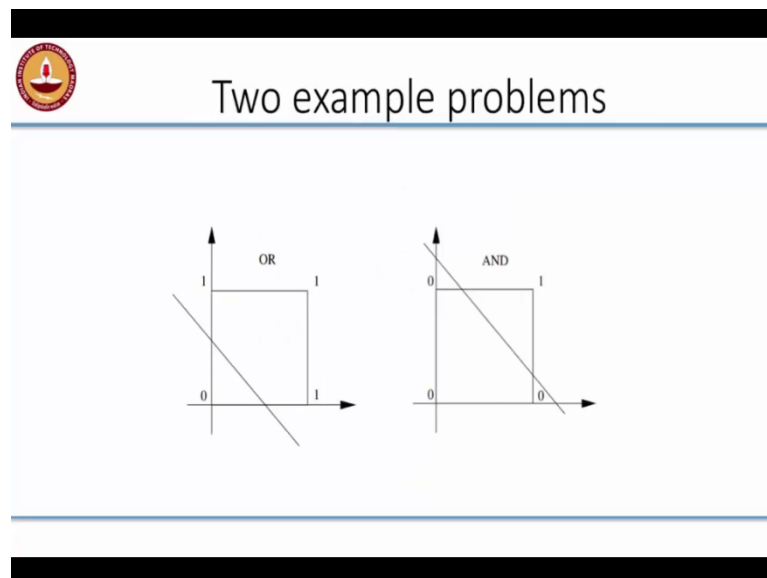


So, let us take a look let us just back and take look at what can of perceptron learning is a

news paper article really true or what are the limits to the perceptron's learning ability. So, here is a very simple perceptron here, so it has a two input variable x_1 and x_2 and that is the threshold of 1 and the weight w_1 is 0.9 and w_2 's 2. So, if you look at it essentially it implements this straight line here, so everything above the straight line this light color regions belong to one class and the dark color regions belong to another class.

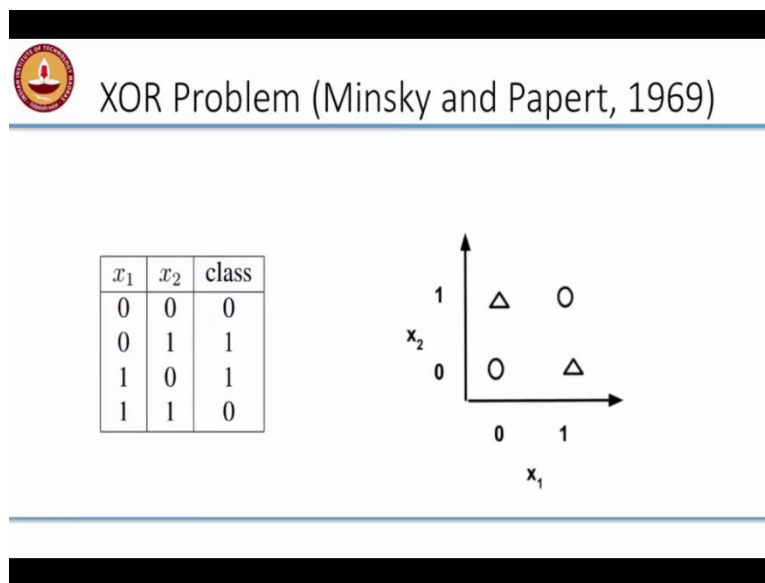
So, we know that these are data which are linearly separable; you saw this in the case with SVM's. So, these are data that are separated by a linear hyper plane or the linear separating surface. So, all data points for which the $w^T x > 1$, will get a class of + 1 all those that evaluate to lesser than 1 and get a class of - 1.

(Refer Slide Time: 13:03)



So, again let us go back and look at the simple logic function that we saw earlier. So, it can implement that OR. So, essentially OR requires you to have a hyper plane and this passing here. So, everything to this side this become + 1 everything to this side become - 1 and likewise you can implement and so you can draw a simple hyper plane. So, everything to this side become + 1 and everything this side becomes - 1 or 0, I mean depending on how you wanted to predict the output.


(Refer Slide Time: 13:31)



And let us look at another one, look at simple problem just like OR and AND the XOR problem. So, Minsky and Papert in 1969 in a famous monograph called the perceptrons showed that well a simple problem like XOR. So, where the truth table is given here is the inputs of the same output is 0, if the inputs are differently output of 1, the simple problem like XOR is not linearly separable, you cannot draw a hyper plane that separates these two classes.

So, forget about walking, forget about talking and doing all those wonderful things that was claimed to newspaper article perceptron's cannot even solve this as simple problem as XOR is essentially says that two things are same, the output is 0, two things are different the output are 1 that we cannot recognize the similarity between this simple inputs like 0's and 1's what kind it do to complex computations. So, once Minsky and Papert showed this, it is a kind of you dampened the research into neural networks for a long time until there was revival much later.

(Refer Slide Time: 14:41)

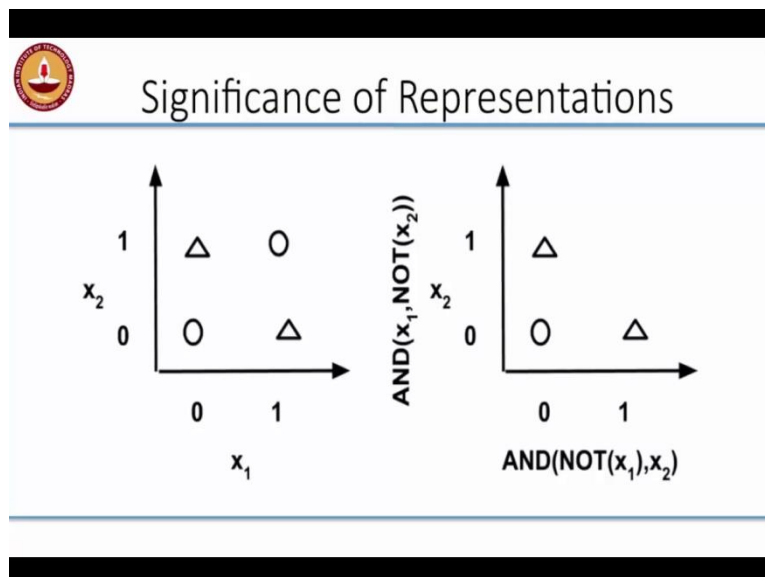


What can a perceptron learn?

- Perceptrons can learn only linear decision boundaries!

So, perceptron's can learn only linear decision boundaries that is the take away message here. So, that is make that is whole idea of neural networks completely useless, because they can learn only linear decision boundaries in case of SVM's we saw that we could get it to do all linear boundaries by going into Kernel expansion this has something similar that we can do here.

(Refer Slide Time: 15:05)



Let us look at how we can change the representations and try to do something more clever. So, if you look at the original problem the XOR problem, so I have my inputs x_1 and I have my input x_2 and now we can see that in this space the problem is not separable. But, let us look to do a simple transformation on my data points, so instead of

looking at x_1 I will define my first variable as NOT x_1 and x_2 and similarly I will define my second variable as x_1 and NOT x_2 .

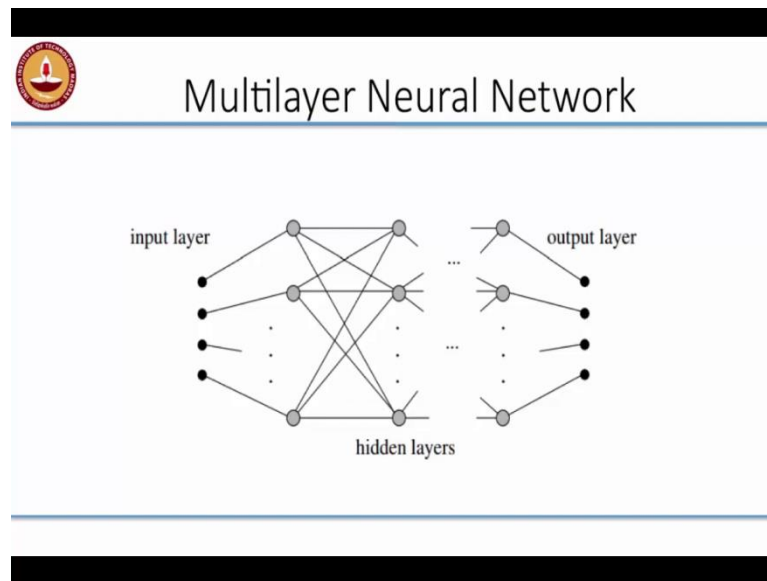
So, if you think about it, so we can now plugging different values of x_1 and x_2 here and see what the outputs will be and then you can see that when x_1 is 0 and x_2 is 0, the output is going to be 0, when x_1 is 1 and x_2 is 0. So, the output here will be x_1 is 1 and x_2 is 0, the output here will again be 0 and x_1 is 1 and x_2 0 the out here will be 1. And we know that 0 1 the output has to be 1, so that we get it here and likewise for the symmetric case this will be the output and so you can see that this is again going to be 1 and when x_2 x_1 x_2 both are 1 again the output will be 0 0 and therefore, this is the resulting point.

Now; obviously, this representation the data points are linearly separable. So, now, the task becomes one of finding the right representation, such that the data becomes linearly separable for the next level, next stage of computation. So, people realized this very quickly, so even though a single perceptron cannot solve complex problems like XOR which are not linearly separable, he could actually stack layers of neural neurons and then have the first layer compute something that is simple.

So, you can always compute NOT of x_1 we saw that earlier and also can be computed by a single neuron. So, you can this get have layers of neuron that exactly compute your features and of NOT x_1 , x_2 and then have another neuron, which takes the output of this neurons combine same together and produces the output that you want. So, people very quickly realize that stacking these kinds of neurons into layers allows you to do more complex computation.

In fact, it is easy to show that stacking these neurons into layers actually builds a universal function representation that learns to a represent any Boolean function, you see a combination of neurons. So, what is a problem, now we know how to solve this more complex problems, why did the research in neural networks pick up again.

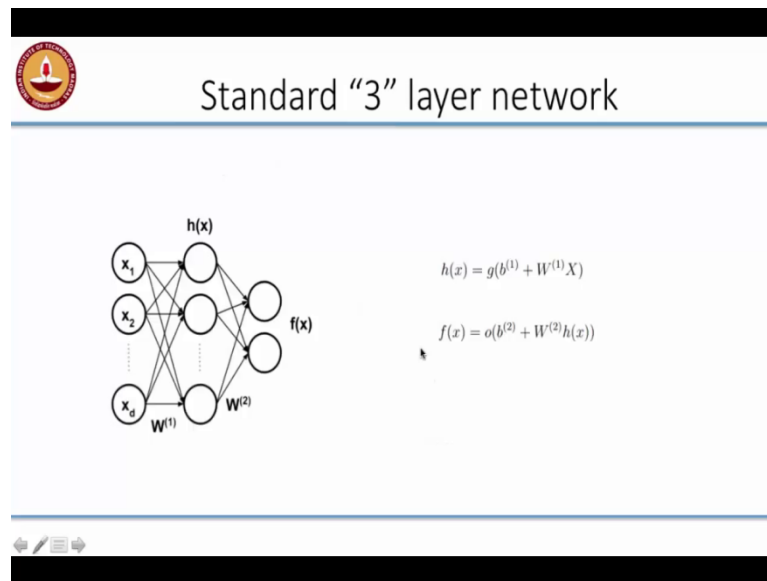
(Refer Slide Time: 18:06)



So, the question here is when I start connecting all of these neurons into layers. How do I find the weights? So, perceptron learning algorithm might no longer work in this case actually does not work in this case and people were struggling to come up with the mechanism for training all these weights. So, you can see that the way of started putting these things into layer. So, that is one input layer and one output layer, so there is one input layer, there is one output layer and in between this you could have many layers of neurons, they are typically called hidden layers because you do not observe their outputs directly.

So, now, we have this many, many hidden layers of weights and it is little hard to find out what this weight should be and so in the mid 80's around 83 an algorithm was proposed called back propagation which allow you to learn the weights of this and we solve this hidden layers.

(Refer Slide Time: 19:15)

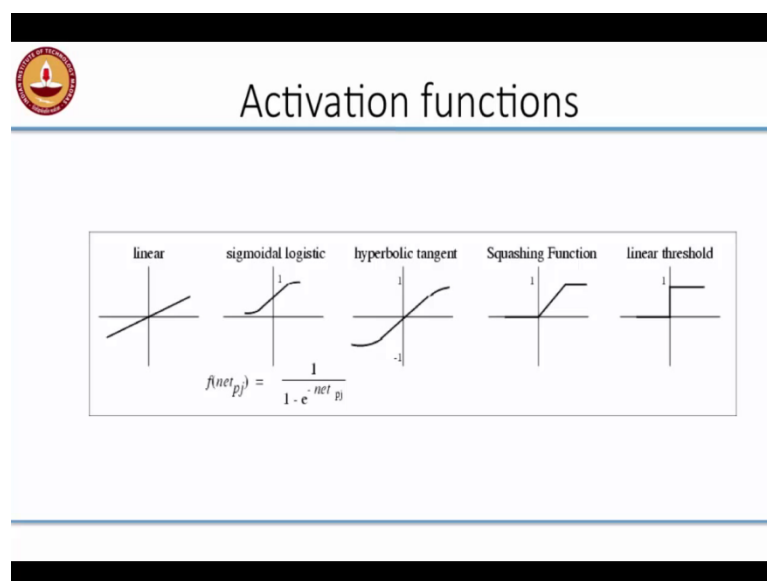


So, for the rest of the presentation, we will be looking at the standard three layer network. So, there is an input layer x_1 to x_d and the output layer which will denote by f of x and one hidden layer of neurons. So, these take the inputs from the input layer do the weighted sum do your thresh holding function and then produce an output and then the neuron and output layer will take all this outputs of the hidden layers take their weighted sum and take the threshold or not and that produce the output, instead of using hard threshold we use a kind of a soft threshold in order to do this competitions this is needed, so that you can derive more efficient training algorithms later.

So, the output of a hidden units and it given by $g(\text{bias})$, this is the θ that we had earlier. So, instead of θ so it is going to call it $o(b^1 + W^1 * x)$ and the output of this will be note by $h(x)$ and the output of the final layer of neurons is given by some function $o(b^2 + W^2 * h(x))$. And so now, the goal here is to figure out what this W^1 and W^2 are going to be. So, this is called the three layer network, even though there are only two sets of weights that we have to learn.

So, the layers here talk about the neurons here, so we use for each input variable we are same that there is separate neuron that is activating the hidden units. So, this is called the standard three layer network structure.

(Refer Slide Time: 20:59)



So, what are the different activation functions you can use? So, we already looked at one which is the threshold function, we can also have just a linear activation function that basically takes the summation of weighted summation of all the inputs and outputs as it is. We can also look at the sigmoidal function, sigmoid logistic function which takes the summation input and then squashes the input. So, that it remains between 0 and 1 and then there is a steep raised somewhere around the threshold. So, that it transitions rapidly from 0 to 1.

When if you are interested in having signed outputs then you can think of using a hyperbolic tangent, where the outputs are going to taxation between - 1 and + 1 and again around the threshold. So, there are parameters at control where the threshold would be and how steep the price would be. So, another transition function some time gives is this squashing function, which is 0 before the threshold and one at a certain distance higher than the threshold and in between you have a, linear approximation adds to the step function, this called the squashing function.

So, typically in most of the neural network architectures that we look at will be looking at either the hyperbolic tangent or this, the logistic sigmoid or the linear activation, because these are different shape and this allows as deriving efficient training algorithms for the same. So, if you are doing a classification problem then the output layer could be the hyperbolic or a logistic sigmoid and if your solving a regression problem, the output neuron could be a in linear neuron. So, that you can do appropriate regression fit.

The hidden layer almost always has to be a non-linear function and where the little bit of what you can show that if the hidden units have a linear activation, like you might as for not have them at all. And what is the function that is implemented is something which can be as well implemented by a single layer of neurons. And the next module we look at how you the exactly find out these weights given the assumption that they are working with the sigmoidal logistic function.

So, the function for the sigmoidal logistic thing is given by $f(\text{net}) = \frac{1}{1+e^{-\text{net}}}$. So, that is the function and look at, given that this is the activation function how we are going to derive the weights of the two layer standard three layer neural network. So, that is in the next class.