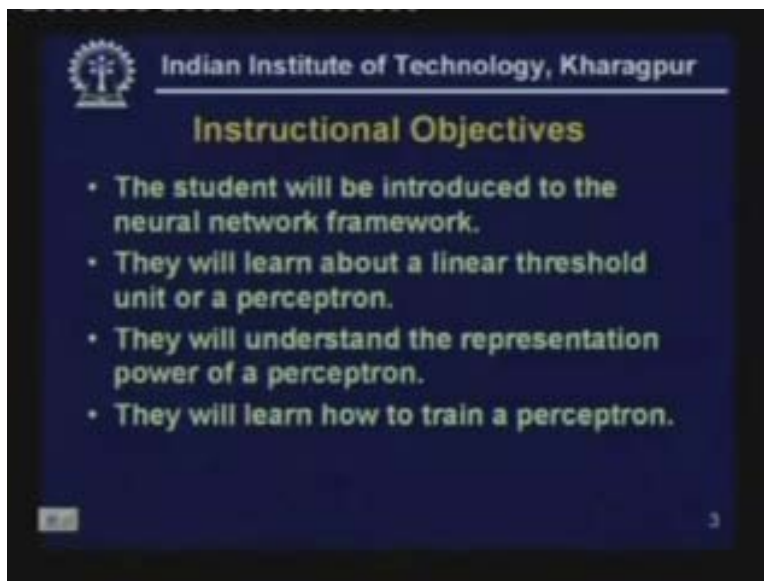


Artificial Intelligence
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture - 36
Learning using Neural Networks - I

Welcome to the next lecture on Artificial Intelligence. Earlier we looked at the paradigm of machine learning and we discussed different learning problems. And specifically we talked about what concept learning is and we mentioned that different algorithms can be used for the concept learning task or for classification task. **In the last two classes we discussed a particular type of learning algorithm** which is induction using decision trees. We looked at algorithms to learn decision trees from data.

Today we are going to look at another model using which we can do concept learning. The model that we will talk about is actually a general model and different variations of it can be used for many other tasks. What we are talking about is the connectionist paradigm which is used and which is more commonly known as neural networks. We will first briefly discuss what a neural network is and then we will see how some types of neural networks can be used for the concept learning task.

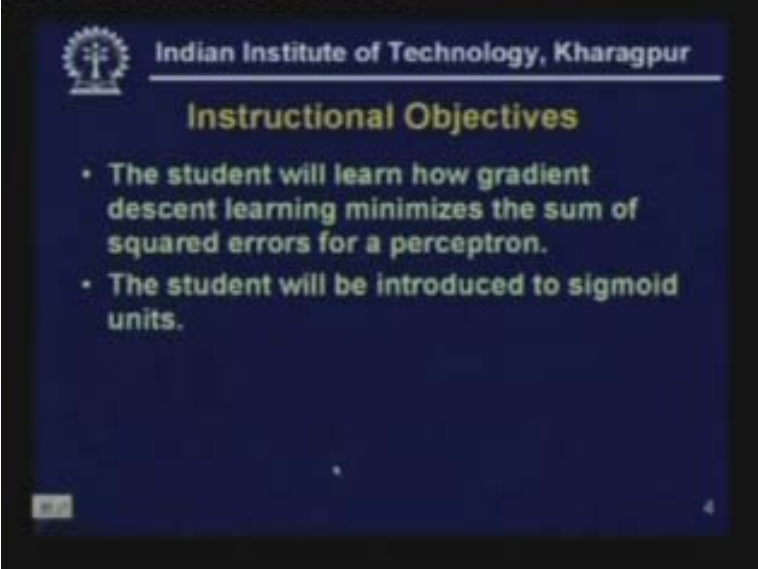
(Refer Slide Time: 02:20)



The instructional objectives of today's lecture are as follows:

The student will be introduced to the neural network framework. They will learn about a linear threshold unit or a perceptron which is a very simple model of neural network. The student will understand the type of functions that can be represented using a perceptron. The student will learn how a perceptron can be trained given some data to fit the training data.

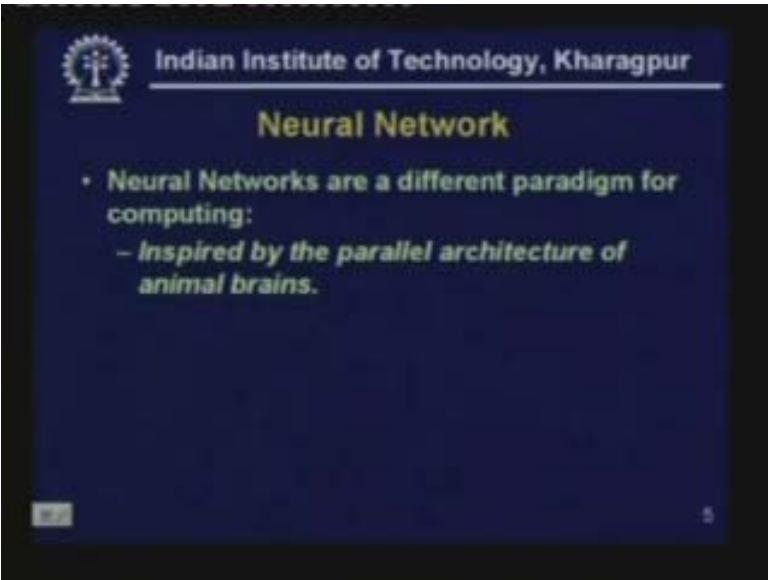
(Refer Slide Time: 03:00)



The slide features the IIT Kharagpur logo in the top left corner. The text is centered and reads: "Indian Institute of Technology, Kharagpur" followed by "Instructional Objectives" in a larger font. Below this, there are two bullet points: "• The student will learn how gradient descent learning minimizes the sum of squared errors for a perceptron." and "• The student will be introduced to sigmoid units." The slide number "4" is in the bottom right corner.

And then they will learn how gradient descent can be applied in order to learn a perceptron to learn a simple linear unit. They will see that the gradient descent learning minimizes the sum of the squared error for a linear unit. And we will also introduce the students to a sigmoid unit which we will use as a basis for multilayered neural network. So, neural networks are actually different paradigm for computing just like the conventional computers that you have studied is based on the Von Neumann framework of computing.

(Refer Slide Time: 03:52)

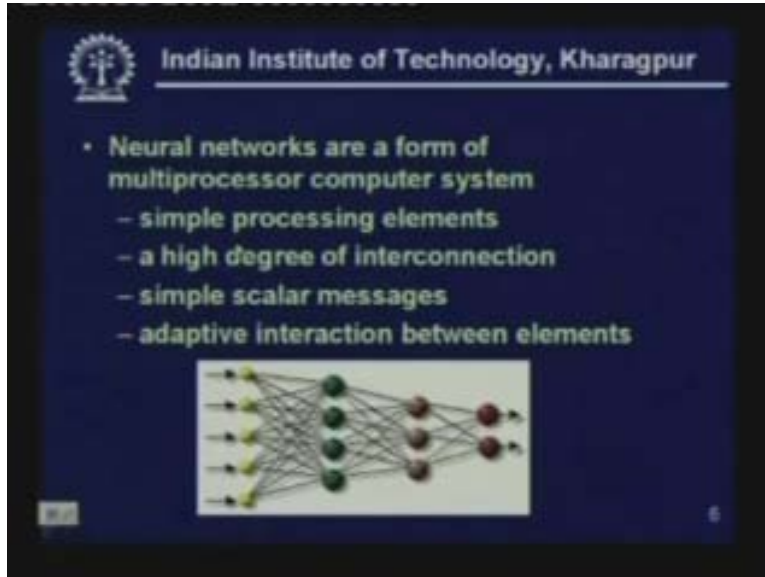


The slide features the IIT Kharagpur logo in the top left corner. The text is centered and reads: "Indian Institute of Technology, Kharagpur" followed by "Neural Network" in a larger font. Below this, there is a bullet point: "• Neural Networks are a different paradigm for computing:" followed by a sub-bullet: "– *Inspired by the parallel architecture of animal brains.*" The slide number "5" is in the bottom right corner.

The neural networks is based one a slightly network different idea of computing. This paradigm has been inspired biologically by trying to emulate the parallel architecture of

human and animal brains. So, a neural network system comprises of many units which work in parallel. These units are individually simple units but together they can be varied to perform more complex tasks.

(Refer Slide Time: 04:39)



Indian Institute of Technology, Kharagpur

- Neural networks are a form of multiprocessor computer system
 - simple processing elements
 - a high degree of interconnection
 - simple scalar messages
 - adaptive interaction between elements

The diagram shows a neural network with three layers of nodes. The first layer has 4 yellow nodes, the second has 3 green nodes, and the third has 3 red nodes. Lines connect nodes between adjacent layers, representing interconnections.

So the main highlights of this framework are:

There are many simple processing elements, there is a high degree of interconnection between these processing units and there are simple messages which have been transferred to this interconnection and the interaction between these different elements is adaptive. This is the essence of the neural network paradigm. Let us look at the history of neural computing. In 1943 McCulloch and Pitts did some work which is usually recognized to be the first work on neural networks.

(Refer Slide Time: 5:38)

Indian Institute of Technology, Kharagpur

History

- 1943- McCulloch & Pitts are generally recognised as the designers of the first neural network
- 1949-First learning rule
- 1969-Minsky & Papert - perceptron limitation - Death of ANN
- 1980's - Re-emergence of ANN - multi-layer networks

In 1949 the first rule for learning neural network was devised. In 1969 Minsky and Papert published a paper which highlighted the computational limitations or representational limitations of a perceptron unit. This leads to a stop of virtual decline in research world in artificial neural networks. Fortunately the 1980s saw a re-emergence of interest in artificial neural network and many researchers came up with more complex architectures in the form of multilayer networks that overcome the limitation of the perceptrons. And today research in the area of neural networks is quite active and they have been used or being used in a variety of application areas.

(Refer Slide Time: 6:50)

Indian Institute of Technology, Kharagpur

Why Neural Networks?

Conventional Computers	Neural Computers
Fast Arithmetic	Handling Noisy Data
Precise algorithmic solutions	Parallelism
	Fault Tolerance
	Adaptation

Now let us very briefly look at the difference in the architecture of conventional computing and neural computers. Conventional computers are known for the fast computations or fast arithmetic the number crunching that they can do and they are very good for delivering precise algorithmic solutions. Precise algorithmic solutions can be programmed in a conventional computing model.

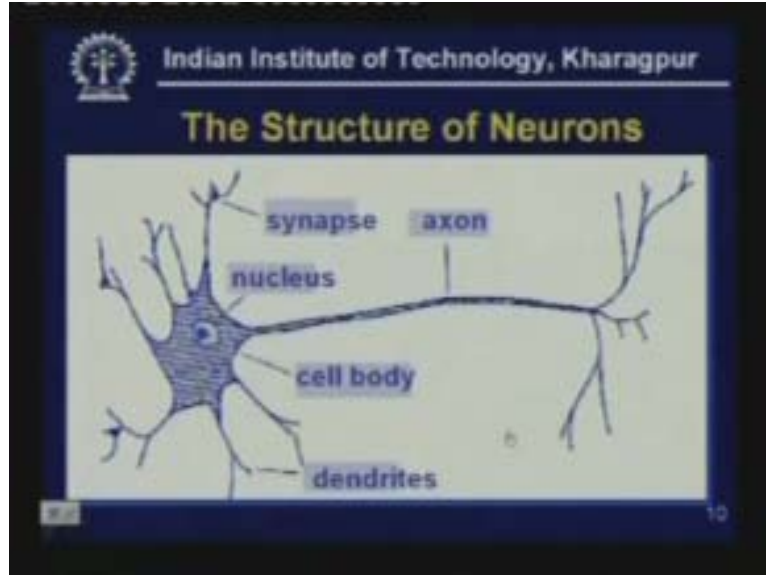
On the other hand, the connection is paradigm and is very good for handling noisy data, for taking into account the massive amount of parallelism between different processors such computation are called fault tolerant and they are adaptive. The connectionist framework has been biologically inspired by the structure of animal or human brains. We all know that humans and animals are extremely intelligent and can do certain tasks remarkably well. And so far sophisticated computers have not been able to come up to the level of human or even animals in performing certain routine tasks. In the beginning lectures we looked at some such tasks as computer vision, image recognition, speech, language understanding and so on.

(Refer Slide Time: 8:37)



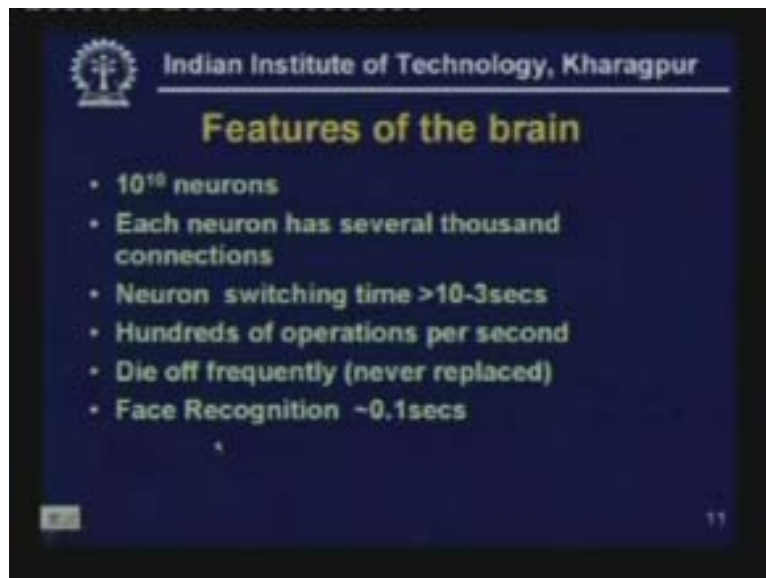
Therefore many **cognatic** scientists and psychologists or any other people have extensively studied the brain in humans and some other animals. But it is really very complex like many other biological structures and functions. It is a very complex unit and it is very difficult to understand the brain completely. Actually the brain is composed of a number of units known as neurons. These neurons are also individually very complex units and together they constitute the brain. Therefore the basic structure of a neuron or nerve cell which occurs in the brain as well as in the other parts of the body is described in this diagram.

(Refer Slide Time: 9:29)



The neuron has a nucleus and a cell body and the tail called axon. There are dendrites which are units that out from the cell body and at the end of the dendrites there is a synapse through which communication takes place between different neurons.

(Refer Slide Time: 10:03)



Some salient features of the human brain:

10^{10} neurons, each neuron has several thousand connections, neuron switching times are of the order of 10^{-3} seconds that is 1 microsecond and neuron can perform hundreds of operations per second, also neurons die off frequently and some of

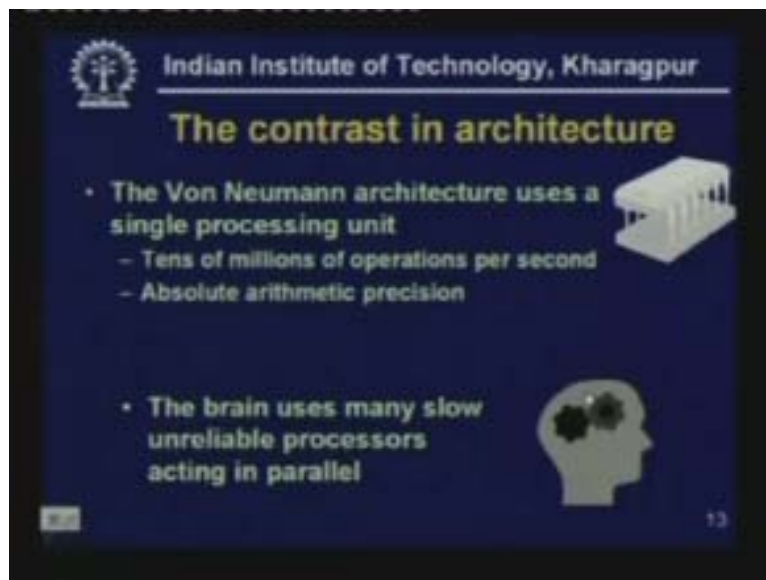
them are never replaced. These are the salient features of human brain. And we all know the sort of tasks that human brain can perform.

For example, we are very good at recognizing faces and typically recognizing faces takes us an average of 0.1 seconds. Now let us try to contrast this with the structure of what we have been able to achieve in modern computer. And we will notice that even though the number of gates or transistors in a modern day computer is large it is less than what you find in a human brain. However, the switching time of the devices that make up a computer are very fast and our processing elements can work faster than the processing units in the brain.

Salient features:

The brain is good for pattern recognition, for association like faces with names, complexity and they can tolerate noise to a large extent. A modern day computer or a machine On the other hand, is good for calculation for precise answers and for being able to apply logic on rational process.

(Refer Slide Time: 12:09)



The Von Neumann architecture used in conventional computing usually comprises of a single processing unit which can perform tens of millions of operations per second and the arithmetic precision is very high. On the other hand, the human brain consists of many slow unreliable processors that act in parallel. And we know that the brain is still capable of doing many complex tasks. The idea between the connectionist paradigm is to try to **emulate** this structure of the brain and to devise a computing paradigm is **wired** by which can perform complex tasks using this sort of brain work.

Neural network comprises of many simple processing units. So we will first study a simple processing unit which we call perceptron and we will discuss what a perceptron can represent and how a perceptron can be learnt. So the unit that we are describing now

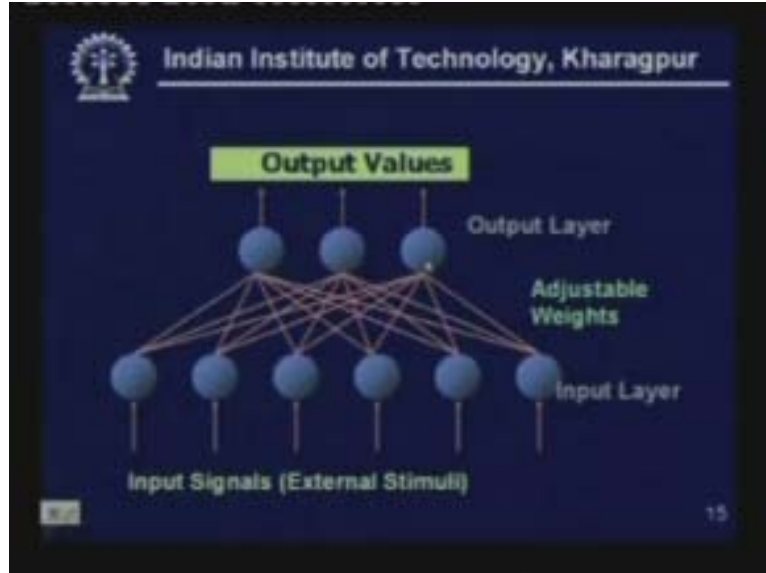
is called a perceptron or a linear threshold unit. In a linear threshold unit the unit can take a number of inputs. And x_1 x_2 up to x_n are the different inputs to this unit. These inputs are connected to this ltu by some arcs and there are weights associated with each of these arcs. For example, this arc has a weight of w_1 , this arc has a weight of w_2 and this arc has a weight of w_n . Now here what this unit does is it finds out the weighted sum of these inputs that is w_1x_1 plus w_2x_2 up to w_nx_n and this unit performs the function of thresholding. So this unit finds a weighted sum of inputs and then it applies the threshold. If the sum is greater than a threshold then the output of the perceptron is 1 otherwise the output of the perceptron is minus 1.

So this summation unit is followed by a thresholding unit. In this thresholding unit the threshold is applied. This threshold can have a special input x_0 which is equal to 1 and associated with it we can have a weight w_0 for which we have to adjust to get the threshold. So this entire unit really computes this function $\sum w_i x_i$ for i varying from 0 to n and if this summation is greater than 0 then the output is 1 otherwise the output is minus 1. So this is the simple linear threshold unit. so this part is the linear part which computes the sum and this part applies a non linear step function on this sum and the output from this unit is either plus 1 or minus 1.

So, typically in a neural network we have inputs. We can have more than one input. These inputs are fed to the network through what is called the input layer. And then each of these things is the output. The outputs are known as the output layer. To compute this output these inputs are fed to the outputs through these connections. In this case there are six inputs and all these six inputs are fed to each output unit and this output unit provides an unselect y_1 . This is the input x_1 , this is x_2 , this is x_3 , this x_4 , this is x_5 and this is x_6 . And corresponding to these six inputs there are the six arcs with feed into the output unit and output of this unit is y_1 . There are weights on these arcs w_1 w_2 w_3 and so on. Now these weights can be adjusted.

Hence in a network of perceptron the system works as follows: the weights on the arcs are changed. So what happens is that we want to adjust the weights on the perceptron so that the function we want to represent can be correctly represented. As in other concept learning tasks we start with the training data and then we find whether the function that we wish to compute at this output is being indeed computed at these output units.

(Refer Slide Time: 18:50)



So we try to find out the correct function is being computed. If the correct function is not being computed we will change the weights so that the correct function can be computed. So what we normally do in training and neural network is we start with an initially arbitrary value of the weights the way and we inspect the data we have got. If the data we got agrees perfectly with the current structure of the perceptron then we are very happy. But if does not then we change the values of the weight so that it represent the data in a better way. Hence this process is called perceptron learning or training the neural network.

(Refer Slide Time: 20:11)

The slide is titled "Perceptron" and features the Indian Institute of Technology, Kharagpur logo and name at the top. The main content includes a bullet point: "• The network adapts as follows: change the weight by an amount proportional to the difference between the desired output and the actual output." Below this, two equations are presented: $w_i = w_i + \Delta w_i$ and $\Delta w_i = \eta (t - o) x_i$. The slide number "16" is visible in the bottom right corner.

Suppose we are given some data or given one example data and according to the data there are some inputs namely x_1, x_2, x_n and corresponding to this input the target output is t , so if our network correctly represents the function then the network for the input x_1, x_2, x_n should give an output of t . But if our network does not exactly represent the function it will give different output like o . Therefore if t and o are identical then we do not change the weights. But if t and o are different then we adjust the weights of the network so that o is closer to t . Therefore for a simple perceptron the general weight training rule is like this; after processing some training examples each weight is on arc is just as follows: w_i new is old w_i plus delta w_i . So to each weight we apply an incremental change of delta w_i .

What is delta w_i ?

We take delta w_i is equal to η times t minus o into x_i . So t minus o is the difference between the target and that is achieved out. If t is equal to o then we do not change the weight delta w_i is equal to 0. But if they are not same then we change delta w_i proportional to t minus o times the weight of the input. And along with it we use this parameter η which is called the learning rate. If the learning rate is high we make a bigger change and if η is small we make a smaller change.

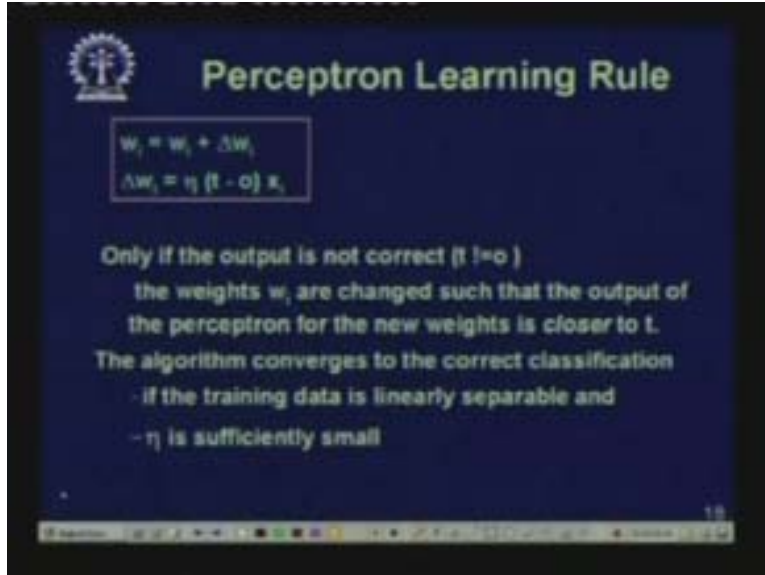
Suppose the target is 1 so for this input the output should have been 1 but o is 0. In that case what we are doing is if t is 1 and o is 0 t minus o is positive and we are increasing delta w_i . If we increase delta w_i what will happen? Since our unit does a summation of $w_i x_i$ and if w_i will increase and if x_i is positive then the final output will increase, the target o was 0 so if the value of the input increases it will go towards 1 if x_i is positive. If x_i is negative then delta w_i will be negative and then it will go towards the negative direction. Therefore what this training rule tries to do is, it tries to push o towards the t towards the target value. Hence the training rule is as follows:

w_i is changed by η t minus o into x_i where t is the target value, o is the perceptron output we have obtained and η is the small constant which is known as the learning rate.

What does this training do?

If t is equal to o then do nothing and if t is not equal to o then the weights w_i are changed such that the output of the perceptrons for the new weights is closer to t .

(Refer Slide Time: 24:14)



The slide features a logo in the top left corner. The title 'Perceptron Learning Rule' is centered at the top. Below the title, two equations are presented in a box: $w_i = w_i + \Delta w_i$ and $\Delta w_i = \eta [t - o] x_i$. The text below explains that weights are updated only when the output is incorrect ($t \neq o$), and that the algorithm converges to the correct classification under two conditions: linear separability of training data and a sufficiently small learning rate η .

Perceptron Learning Rule

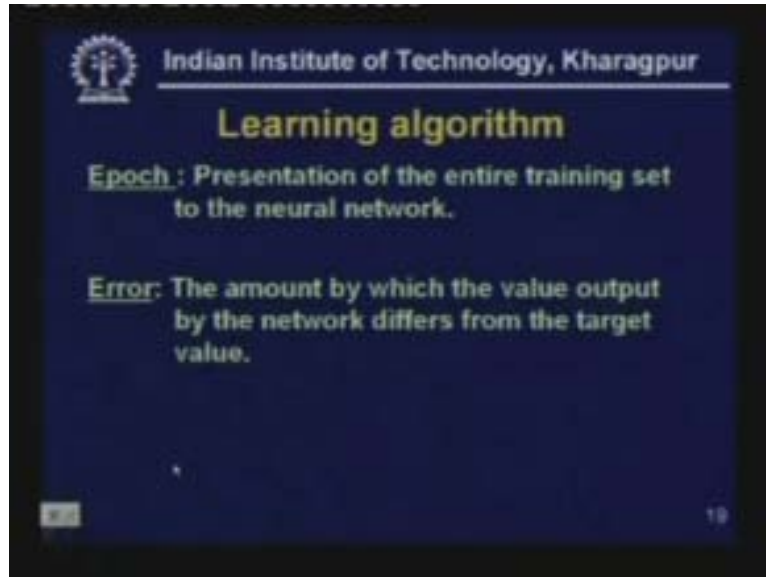
$$w_i = w_i + \Delta w_i$$
$$\Delta w_i = \eta [t - o] x_i$$

Only if the output is not correct ($t \neq o$)
the weights w_i are changed such that the output of
the perceptron for the new weights is closer to t .

The algorithm converges to the correct classification
- if the training data is linearly separable and
- η is sufficiently small

This algorithm converges to the correct classification under the following conditions: If the training data is linearly separable and if the value of the eta is sufficiently small. So the basic idea is that some functions can be represented by a perceptron. And we will see that only linearly separable functions can be represented by perceptron. If the function is not representable by the perceptron then this algorithm may not give you the correct result of the training. But if the function can be represented that is if it is linearly separable then if the learning rate we use is sufficiently small then this process converges. And with initial arbitrary value of the weights by applying the training rule over the data that we have seen we will be able to get finally a weight vector which is able to represent the correct function.

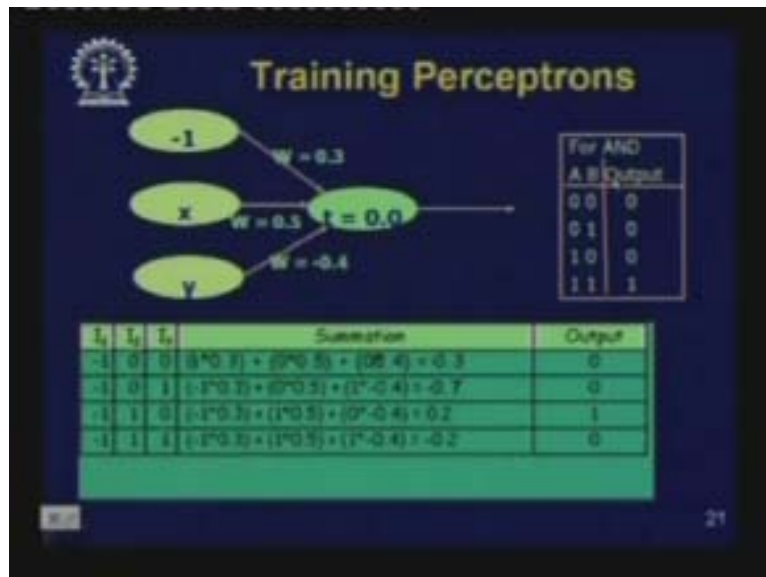
(Refer Slide Time: 25:05)



The basic learning algorithm is as follows:

The learning algorithm works in several phases, several epochs. In each epoch the entire training set is presented to the neural network. Based on the entire training set the error is computed and based on the error the weights are updated. And the error is the amount by which what is the output by the network differs from the target.

(Refer Slide Time: 26:15)



Suppose we have this perceptron structure where x and y are the two inputs and this is the output and the target is 0.0. Now we initialize the weights as random values. Suppose w_0 is equal to 0.3, w_1 is 0.5 and w_2 is minus 0.4. Now suppose the function that we are trying

to learn is the AND function for which the truth table is given here; if a is 0 and b is 0 then the output is 0 and if a is 0 and b is 1 then the output is 0 and only if a is 1 and b is 1 then the output is 1 and in all other cases it is 0. Now, given this network we have to see what the error of the network is for this which respect to this table. For example, if a is 0 and b is 0 then we can try to find out what would be the output. For a is 0 b is 0 what we get is, these links do not get contribute anything but only this link contributes so we get output as minus 0.3 and the output should have been 0. For a is 0 b is 1 we get an output of minus 0.7 and the output should have been 0, for a is 1 and b is 0 we get is 0.2 and the output should have been 1, for a is 1 b is 1 the output is minus 0.1 the output should have been 0. So this is the current error we have on this perceptron and what we will do is adjust the weights using the weight training rule $\Delta w_i = \eta t - o x_i$ is to adjust the weights.

Now, in order to analyze this training rule theoretically what we will look at a unit which is slightly different from the thresholding unit that we have looked at. So what we can do is, we will look at a simple linear unit which performs the summation but does not perform the threshold. So we use a simple linear unit and we want to find out the weight vector for which the linear unit is optimum. The linear unit also will give a value. The thresholding forces here forces the value to be minus 1 or plus 1. So what we will do instead is we will take a linear unit and we will require it to exactly be 0 under the zero conditions, 1 under condition one which we can use. For the linear unit the output is given by w_0 plus w_1x_1 up to w_nx_n . Our task is to learn the value of weights so that the error is minimized.

(Refer Slide Time: 30:03)

Indian Institute of Technology, Kharagpur

Gradient Descent

Consider simple linear units

$$O = w_0 + w_1x_1 + \dots + w_nx_n$$

Learn w_i 's that minimize the squared error

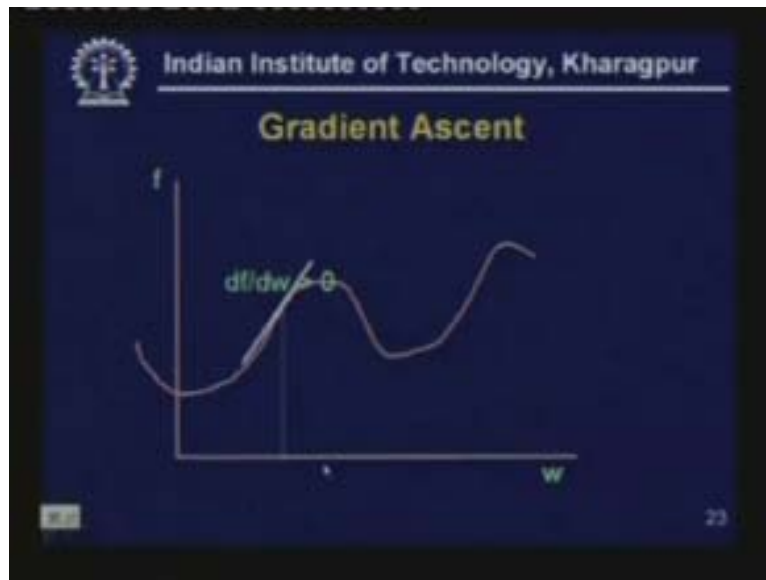
$$E(w) = \frac{1}{2} \sum_{d=0} (t_d - o_d)^2$$

22

Definition of weight: This is the squared error which is a popular measure of error. It is the error with respect to the current value of the weight vector. The weight values are the ones which we are trying to adjust. For the given values of the weights we will have given value of the error and as we change the weights the error function will also change.

So error is a function of the weights. And we use the following definition of error: e_w is half of sum over all training examples $(t_d - o_d)$ whole square. So t_d is the d^{th} is the particular training example it is a target value and o_d is the output for that input by our network. So $t_d - o_d$ is 0 means the network agrees with the example and if $t_d - o_d$ is not 0 then it does not agree. So the summation of these $(t_d - o_d)$ whole square times half is the error function we will use and it is a very common function.

(Refer Slide Time: 32:27)



Now what we will do is inspired by gradient ascent. Suppose we have a curve and this curve is a function of w , if we take a point on the curve and if we find the slope of that point the slope gives the direction in which the curve is moving. Now if we move towards the direction of the slope a little bit we will be going towards the top of the curve. And at the extremum points the slope is 0. If the slope df/dw is greater than 0 that means if we follow the curve we will move towards the peak of the curve. If df/dw is equal to 0 we have reached the extremum position of the curve.

Now our objective is, we are given an error function on the weights and we have to adjust the weight values in order to go to a region where the error is at minimum. Therefore instead of doing gradient ascent we will try to do gradient descent on the error curve and our objective will be to start with the point and then move in the negative direction to the slope until we reach a point where the slope is equal to 0, that is we reach the minimum of the error function. The error depends on the weight vector.

Suppose we have a very simple system or simple learning problem where there are only two weights w_0 and w_1 , now the error function can be given as $\frac{1}{2} \sum (t_k - o_k)^2$ whole square and if we have o_k as a linear unit then we have a quadratic error function. And this quadratic function has the shape of a parabola. The characteristic of a parabola is that it has a unique minimum. When we start we have a particular set of weight values. Suppose this is our current weight value we want to find that combination of weights for

which the error is minimum. So we want to reach this combination of weight values and we have started from this place in the error service.

So what we will do is, we will compute the gradient at this point of the error surface and we will take a small step in the negative direction of the gradient so that we can descent the error surface and we will go on descending the error surface until we reach closer to the global minimum. And finally we might be able to reach the global minimum. For this surface because it is a parabolic surface there is only one minimum which we can reach. So the error on the weights is given by $\frac{1}{2} \sum (t_k - o_k)^2$.

(Refer Slide Time: 36:37)

Indian Institute of Technology, Kharagpur

Gradient Descent

$$E[\vec{w}] = \frac{1}{2} \sum_{k=0}^n (t_k - o_k)^2$$

Gradient: $\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$

Training rule: $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

25

And the gradient of this error function is, we can find the partial derivative of this error function with respect to each weight value. So $\text{grad } E[\vec{w}]$ can be written as $\frac{\partial E}{\partial w_0}$, $\frac{\partial E}{\partial w_1}$ and $\frac{\partial E}{\partial w_n}$. So we have the error slope along every dimension which gives us the gradient vector. Now our training rule will be to change each weight towards the negative direction of the gradient vector so that we can descent the error surface. Here \vec{w} is the weight vector so $\Delta \vec{w}$ is equal to $-\eta \text{grad } E[\vec{w}]$. So $\text{grad } E[\vec{w}]$ is this quantity which is the partial derivative of the error vector. So, each individual w_i is changed as follows: Δw_i is equal to $-\eta \frac{\partial E}{\partial w_i}$ that is the partial derivative of E with respect to w_i . So we have to find out what is Δw_i for the linear unit that we are considering.

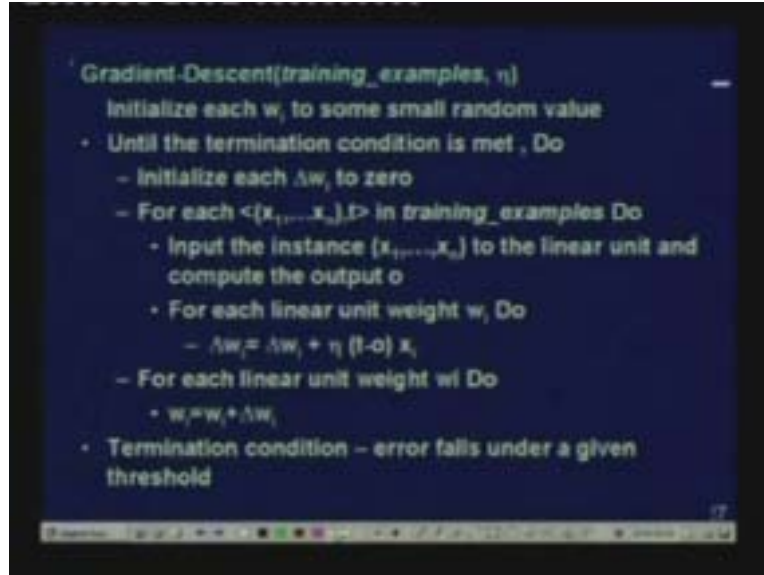
(Refer Slide Time: 36:58)

The slide features a logo in the top left corner and the title "Gradient Descent" in yellow text. Below the title, it states "Train the w_j 's to minimize the squared error" and provides the error function $E[w_1, \dots, w_n] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$. It then defines the gradient as a vector $\nabla E[\mathbf{w}] = [\partial E / \partial w_1, \dots, \partial E / \partial w_n]$. The update rule is given as $\Delta \mathbf{w} = -\eta \nabla E[\mathbf{w}]$. The derivation for the partial derivative of the error with respect to a weight w_i is shown as follows: $\Delta w_i = -\eta \partial E / \partial w_i = -\eta \partial / \partial w_i \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = -\eta \partial / \partial w_i \frac{1}{2} \sum_{d \in D} (t_d - \sum_j w_j x_j)^2 = -\eta \sum_{d \in D} (t_d - o_d) (-x_i)$. A small number "28" is visible in the bottom right corner of the slide.

Hence, in gradient descent we are trying to train the w_i 's to minimize the squared error which is given by $\frac{1}{2}$ sigma over all training examples $(t_d - o_d)$ whole square. As we saw in the last slide $\text{grad } E[\mathbf{w}]$ is a vector of the partial derivatives with respect to the weights and Δw_i is equal to minus eta del e by del w_i . Now let us try to compute del e del w_i for the error function that we have chosen. Now what is e? e is nothing but half sigma $t_d - o_d$ whole square. Now what is o_d ? o_d is nothing but sigma $w_j x_j$. So we have to find the partial derivative with respect to w_i from this function $\frac{1}{2}$ sigma $(t_d - \sum_j w_j x_j)$ whole square.

Now you know that the only portion of this function that depends on w_i is this portion. The $(t_d - \sum_j w_j x_j)$ whole square when we are taking a particular w_i only corresponding to this w_i this term is the only important term. So this derivative can be written as two times $t_d - o_d$ into minus x_i so here the derivative is minus x_i change of variables, we have two times this function $(t_d - \sum_j w_j x_j)$ whole square which is two times $t_d - o_d$ into minus x_i into $\frac{1}{2}$ which gives us eta sigma by d $t_d - o_d$ into minus x_i the negative of this. Therefore to sum it up we have the following algorithm for performing gradient descent on a linear unit given a set of training examples D and where we have n inputs. So the algorithm proceeds as follows:

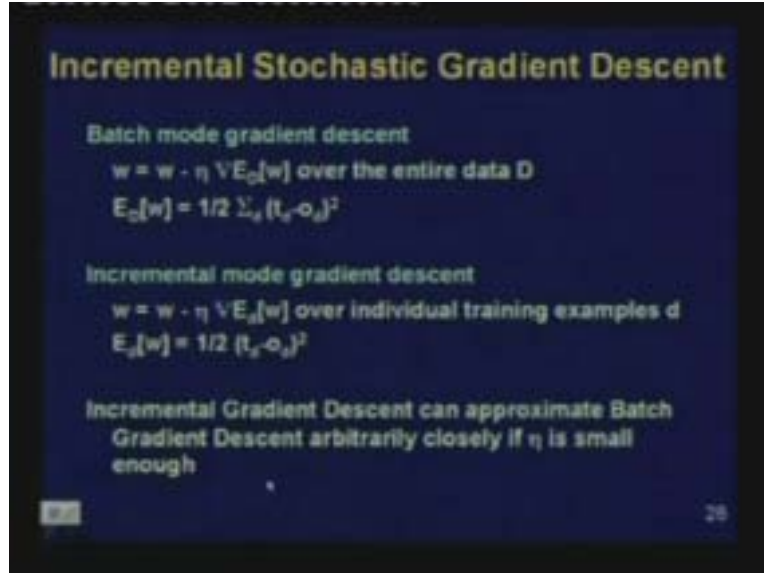
(Refer Slide Time: 39:26)



We initialize each w_i to a small random value. Then we perform the following loop. Until the termination condition is met we do the following: initially we put delta w_i as 0 that is we reset the value of delta w_i . Then we pick up each training example from D . So each training example comprises of inputs x_1 , x_2 and x_n and the target value t . For each of these training examples what we do is we give this input to the linear unit and find the computed output o . The output o is simply $\sigma \sum w_i x_i$. Now, for each linear weight what we do is modify **delta w_i as eeta times** t minus o times x_i and then we say w_i is w_i plus delta w_i this is following a method that we found out that delta w_i should be eeta into t minus o times x_i .

We continue doing this until termination. In the termination condition we terminate when the error falls under a given threshold. So what we are doing is we are looking at all the training examples. For each training example what we are going to do is we are going to find out the error with respect to that training example. And we do it for all the training examples. So we basically finding the sum of this term over all the training examples that is why we have to do, so delta w_i is previous delta w_i plus eeta times e minus o times x_i . And before we have started this we have put delta w_i 0. So what we are doing is basically we are finding out the sigma over all training examples eeta t minus o x_i . So we are doing a version gradient descent which is called a batch gradient descent algorithm where we start with a particular vector we process all the training examples we find the cumulative error and accordingly we change the rates and again we continue this process and stop when the error is accepted.

(Refer Slide Time: 42:19)

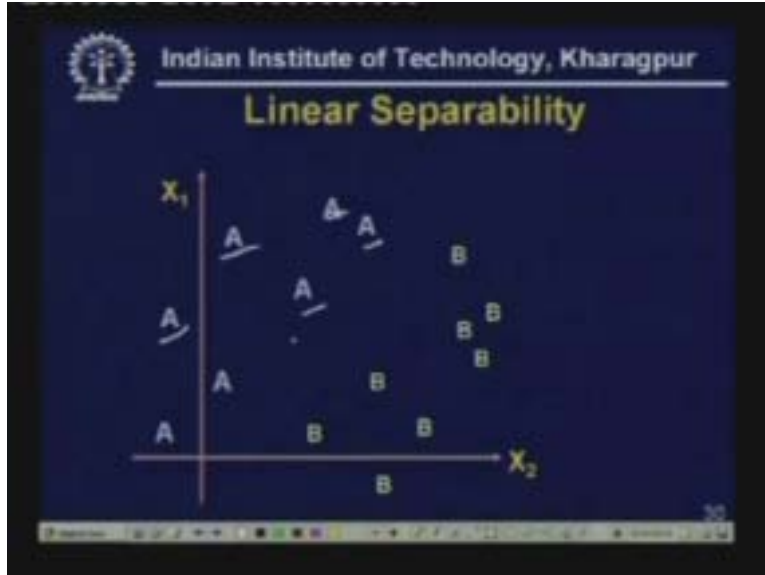


This version of gradient descent this algorithm is called the batch mode gradient descent where the weights take care of the error over all the training examples. However this is expensive to apply after processing all the training example once we are able to adjust the weight values once and therefore this process can take a very long time. So what people often use instead is an incremental version of the gradient descent algorithm where for each example that is processed the weight vector is changed. And this incremental gradient descent is expected to be much faster than the batch mode gradient descent.

In incremental gradient descent what we do is we modify the weight vector by processing a single training example. So we look at a single training example with respect to how we modify the weights, we look at one more training example then we modify the weights and so on. So this process is much faster because after every training example we modify the weights. Therefore the rate of change of weights is much faster. And the good news is that incremental gradient descent is an approximation to batch gradient descent. And if we keep the value of eta small then the incremental gradient descent converges to some value which is very close to what you will get by using batch gradient descent and this is something we usually use in practice.

Type of functions that can be represented by a linear unit or by a linear threshold unit: What we really want to know is what a decision boundary is. The perceptron will basically try to separate the positive examples from the negative examples. And in order to do that we use the geometric concept of decision boundary. So the decision boundary separates the positive from the negative example. For example, suppose we have these two features x_1 and x_2 and we have examples which belong to two different classes, these belong to a and these belong to b and we want to separate a's and b's.

(Refer Slide Time: 42:19)

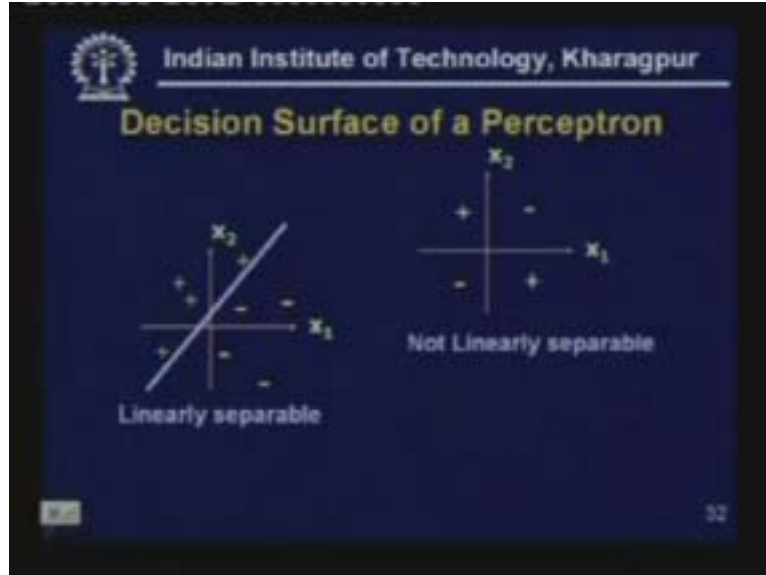


We can have the decision surface so that a's are on one side of the decision surface and b's are on the other side. Now in this two dimensional case we can draw a straight line separating a's from b's. In all cases we will not be able to get a straight line separating a's and b's. But in this case we can get and we say that this function is linearly separated.

What if we have more than two input features?

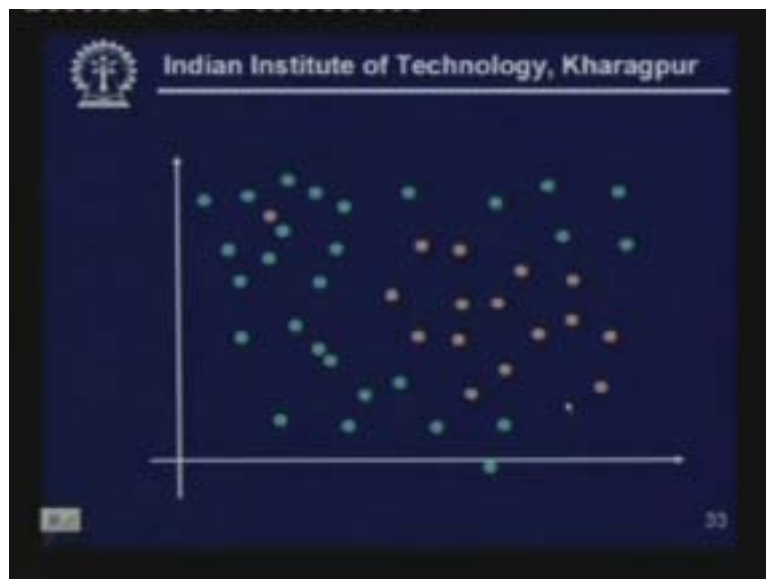
In that case if we have n input features and if we can have a n minus 1 dimensional hyper plane that separates the two classes and then also we will say that the function is linearly separable. The perceptron learns a function of the form w_1x_1, w_2x_2, w_nx_n so this is a linear function which defines a hyper plane.

(Refer Slide Time: 46:29)



So a perceptron can only successfully separate the classes between which there is a linear separator. Therefore this figure is an example of points which are linearly separable. In this slide the green points are separated from the yellow points by this line. Now here we have another example where we have two positive points here and two negative points here. And you can see that we cannot find a plane or find a straight line which separates the positive points from the negative points and such a straight line does not exist. So we say that this training set is not linearly separable. This training set is linearly separable because there exists this line that separates the positive points from the negative points.

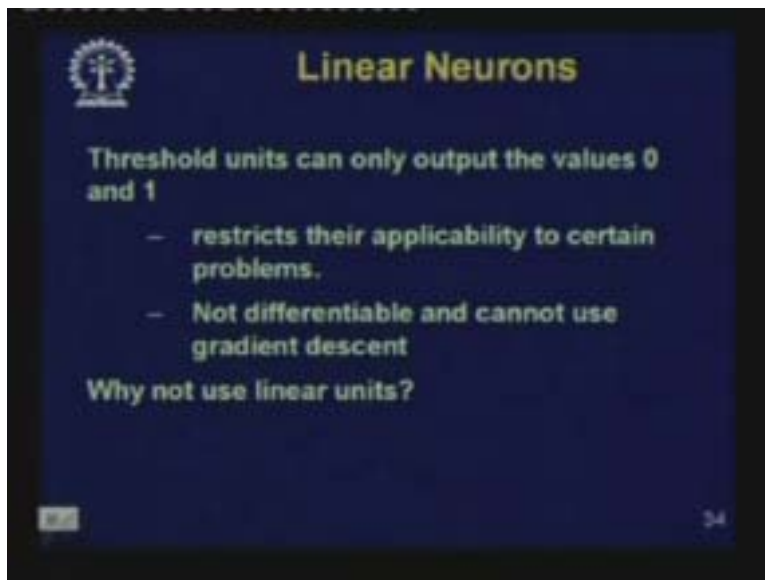
(Refer Slide Time: 47:42)



This is an example of a function which is not linearly separable. Now if you try to learn a function using a perceptron you can find that no straight line will be able to perfectly separate the positive points from the negative points that is the pink points from the green points. So you need a more complex decision surface to separate the positive points from the negative points. Perhaps this decision surface separates most of the pink points from the green point but there is something left over so you also need to incorporate this. So you need more complex decision surfaces to separate this problem.

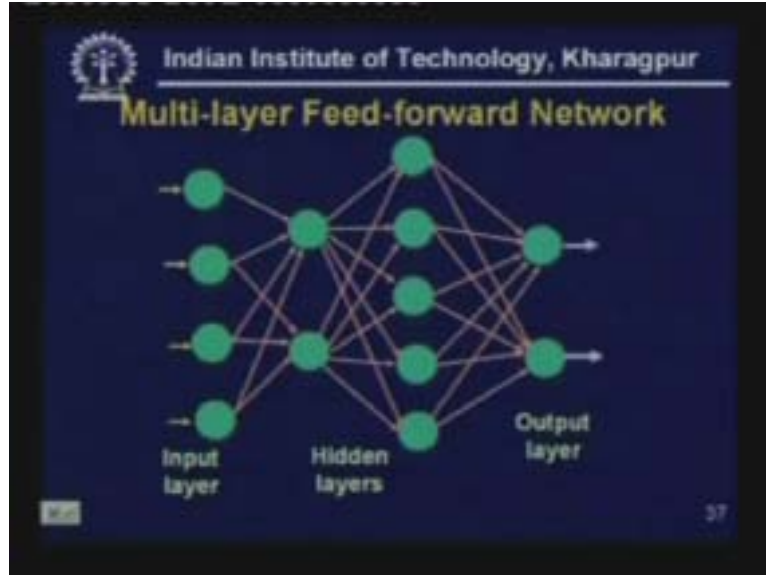
We need to look into neural paradigms where we can deal with such complex functions. Now, when we have threshold units we cannot do gradient descents because thresholding is a discontinuous function. This function the step function is not differentiable and we did the differentiation the derivative finding in order to find the slope. So, instead of dealing with threshold units we can look at linear units.

(Refer Slide Time: 49:12)



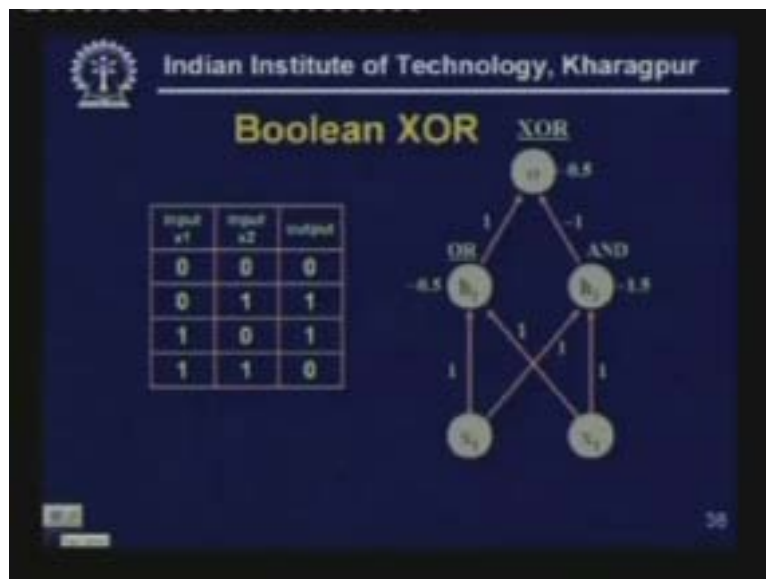
But if we use linear units a single linear unit can only give us linear decision surfaces not more complex decision surfaces. Now what we can do is we can think of having multilayer neural network so that we can cascade these different layer and we can build more complex networks. But if we make more complex networks out of linear units you can easily see that the result is again another linear network. So a linear function is not able to represent all types of functions. So what we want to do is we want to be able to represent more complex functions. And in order to do this we will be using multilayer neural networks. But we can not use linear units because that will not give us the power and we do not want to use thresholding units. Using thresholding units in multiple layers we will see that we can represent more complex functions but thresholding units are difficult to learn because we cannot use the trick of gradient descent. Therefore what we will do is we will try to look at other types of functions which are non linear functions but which are differentiable and using which we can represent more complex functions.

(Refer Slide Time: 51:01)



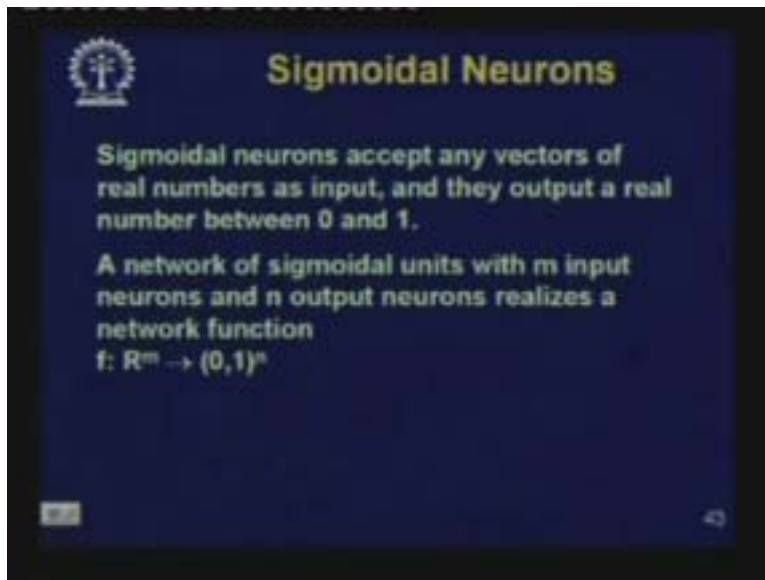
Multilayer neural networks: In a multilayer neural network this is the input layer and this is the output and between the input and the output we put other units where these units correspond to the hidden units. In a layered neural network we will put these units in layers such that the output from one layer feeds as the input of the next layer. This is an example of a layered network where the input goes through several layers and then goes through output. Now we will call this intermediate layers as the hidden layers because they are not part of what is the output. Now we will see that a layered network is able to represent the XOR function which you cannot represent by a perceptron.

(Refer Slide Time: 51:01)



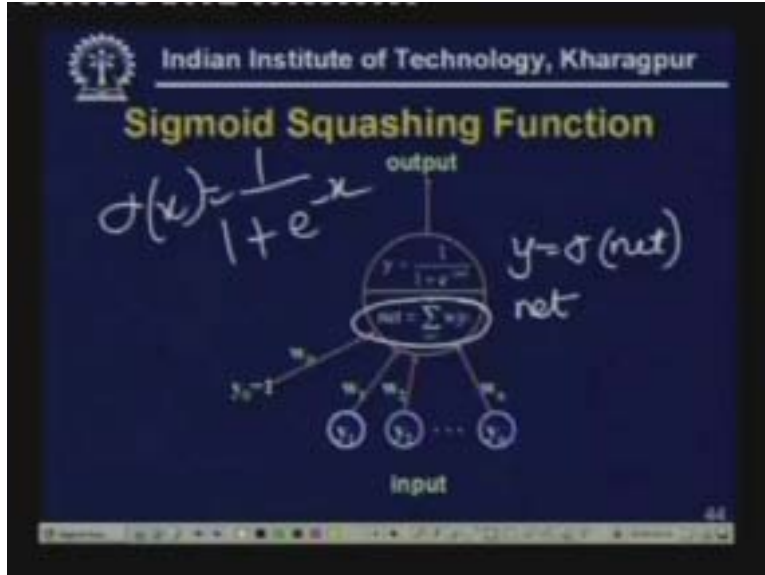
So, if you just think of a XOR function suppose you have two inputs x_1 and x_2 the XOR function has the following characteristic. If x_1 is 1 or x_2 is 0 then it is plus, if x_2 is 0 and x_1 is 1 then also it is plus, if both are 1 it is minus and if both are 0 then also it is minus and there is no line separating the pluses and the minuses. The XOR function can be represented by a two layered network as follows: The OR and the AND function can be represented by a simple perceptron. Find the neural networks that can represent the AND and OR function. Hence this layer can find the OR of x_1 and x_2 and can find the AND of x_1 and x_2 and this can compute the XOR function. So, the XOR function can be computed by a two layered network using three computing units. Thresholding units are not amenable to differentiation, linear units do not give us the power and we are going to look at non linear functions which are differentiable. And the sort of functions we will look at are the sigmoid function.

(Refer Slide Time: 54:07)



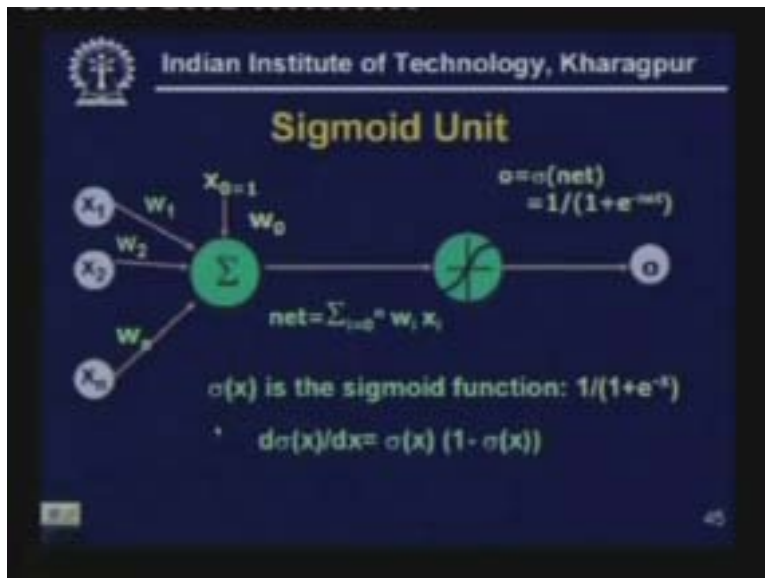
The sigmoid function has a shape as follows: So sigmoid function is a S shaped function which is somewhat close to a thresholding function and it gives us a value between 0 and 1 and we will be using a network of sigmoidal units as our computing units. So, in a sigmoid function we have these inputs then there is first a summation of the inputs and then we apply the sigmoid function the sigmoid function of x is nothing but 1 by 1 plus e minus x . So, the output of this summation unit is net and the sigmoid function we find y is equal to 1 by 1 plus e minus net that is y is equal to 1 by 1 plus e minus net.

(Refer Slide Time: 55:19)



This is also represented here as follows: We have this summation unit which is similar to linear threshold unit and then we apply the sigmoid function so output is sigma of net, net is sigma $w_i x_i$ and therefore output is $1 / (1 + e^{-\text{net}})$.

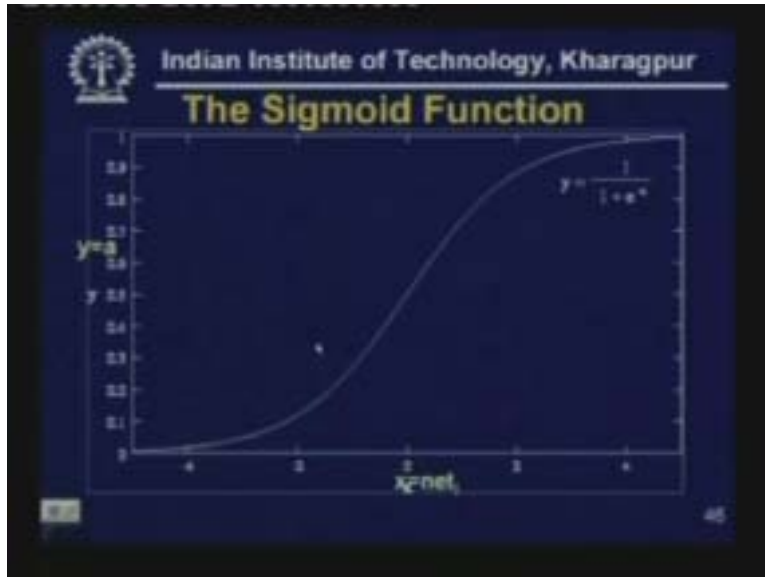
(Refer Slide Time: 55:27)



The sigmoid function as we said is differentiable, the shape is similar to the thresholding function a little bit and also sigmoid functions are easy to manipulate because these $\frac{d \sigma(x)}{dx}$ gives us, so if you do the differentiation you will see $\frac{d \sigma(x)}{dx}$ is nothing but $\sigma(x) (1 - \sigma(x))$. This makes it very easy to compute the differentiation of

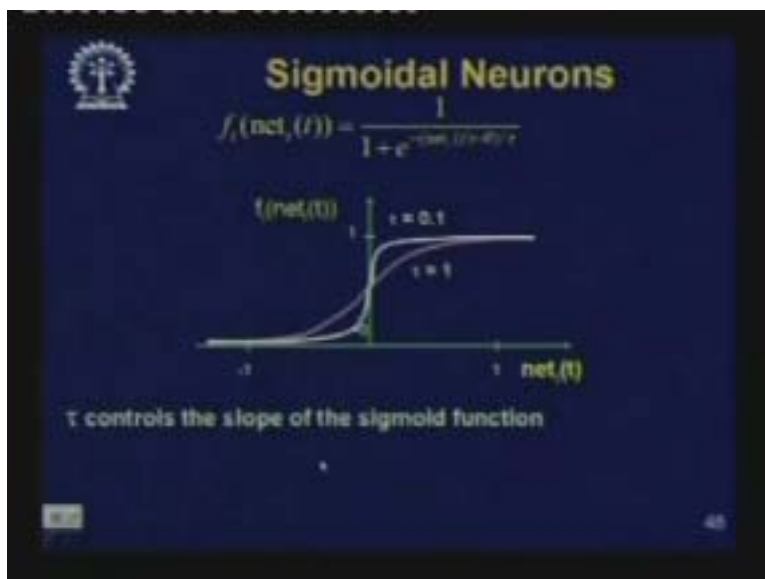
the sigmoid function. Later we will see how multilayered neural network using the sigmoid units can be learnt.

(Refer Slide Time: 56:02)



So this is the example of sigmoid function. The sigmoid function is y is equal to 1 by 1 plus e ^{minus x} . Now if we add another factor here k then if we vary, this function corresponds to k is equal to 1 but if we vary the value of k we will get other functions as t per than the sigmoid function. So, for k is equal to 1 we get a function whose shape is this pink curve but for other values of k is equal to 0.1 we get this curve. So, for different values of k we can find different shapes of the sigmoid curve.

(Refer Slide Time: 57:33)

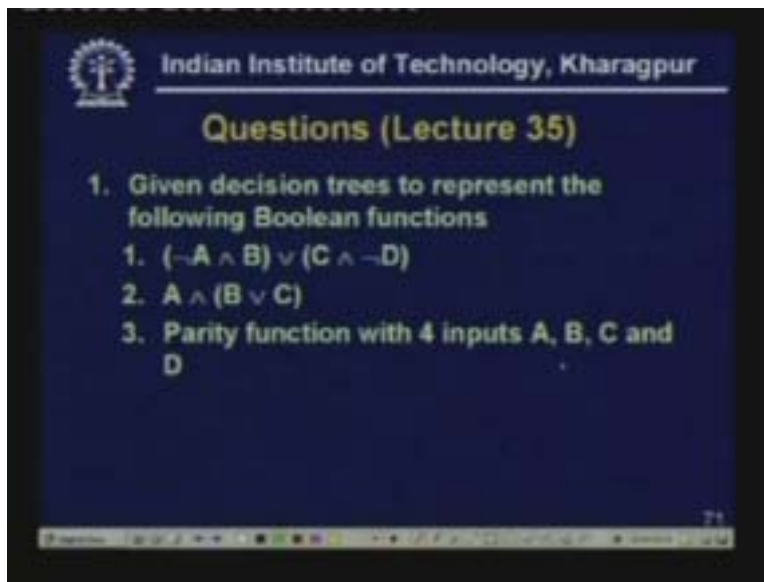


Later on let us learn how to apply the **sigmoid routine**.

Some questions:

In lecture 35 we asked you to represent these functions using decision tree. **These things are very easy to represent using a decision tree**. Suppose for example NOT a AND b OR c AND NOT d so to represented by decision tree what you can do is we can initially test on b and if b is true then we test on a and if a is false then the answer is positive otherwise the answer is negative.

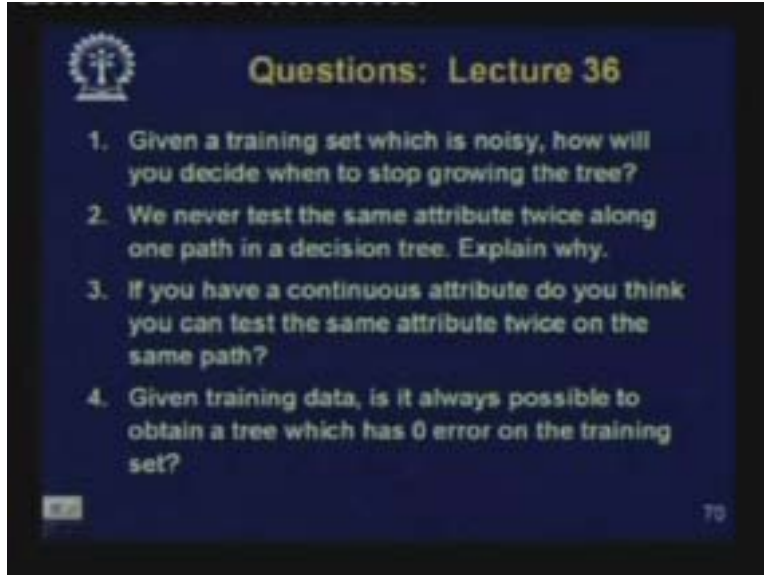
(Refer Slide Time: 58:27)



If b is false then we test on c, if c is true then we test on d, if d is false it is positive and so on so we can build the decision tree corresponding to this function or this function. Take the same functions and try to represent them using neural network initially using a perceptron and if that fails try using a more complex network of two layers.

Parity function of A B C D: The decision tree will not be small it will be actually a full decision tree of four levels where we have to list out all the paths involving A B C D so you start with any of the attributes like a and the next level you check b and c then d you draw the full decision tree and you find out the leaves which will be positive and the leaves which will be negative. So this decision tree is going to be quite large.

(Refer Slide Time: 1:27)



Questions: Lecture 36

1. Given a training set which is noisy, how will you decide when to stop growing the tree?
2. We never test the same attribute twice along one path in a decision tree. Explain why.
3. If you have a continuous attribute do you think you can test the same attribute twice on the same path?
4. Given training data, is it always possible to obtain a tree which has 0 error on the training set?

79

Questions on lecture 36:

The first question was, given a training set how will you decide when to stop growing the tree.