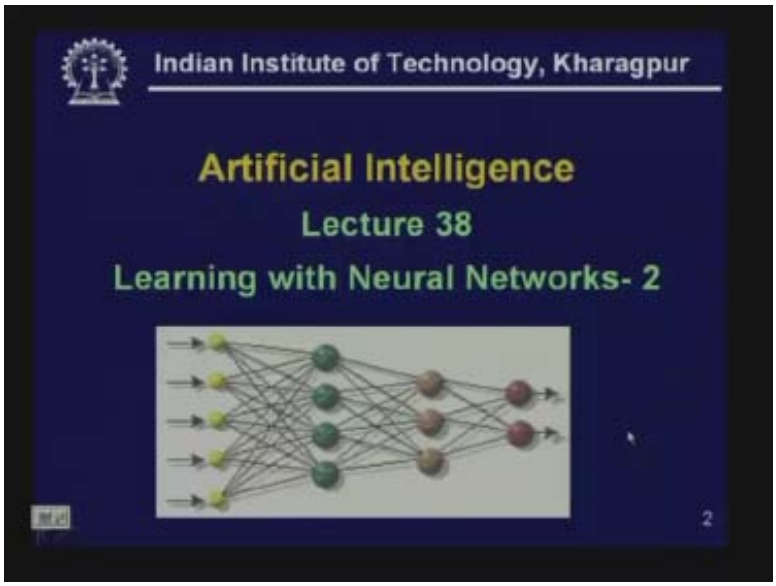
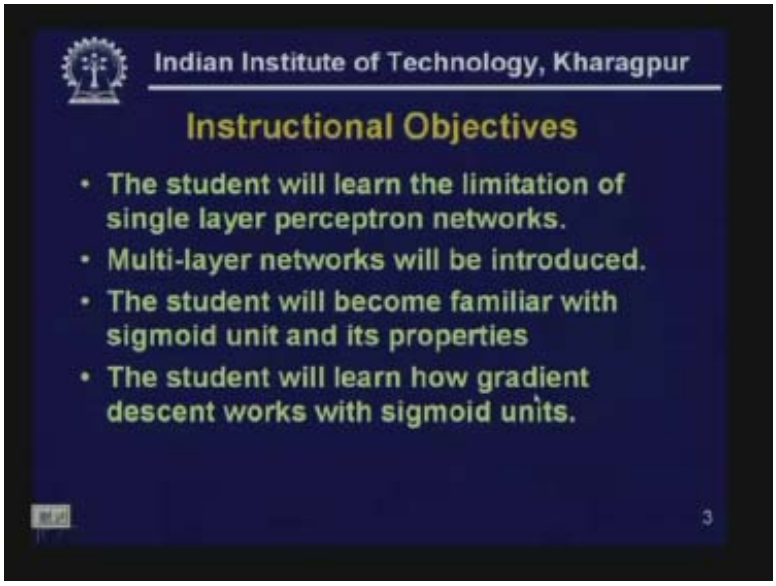
Artificial Intelligence
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture - 37
Learning Using Neural Networks - II

Welcome, today we have a second lecture on neural networks. In the last class we had an introduction to neural network and we discussed linear threshold units or perceptrons and we looked at how to train neural networks using linear threshold units.

(Refer Slide Time: 1:13)

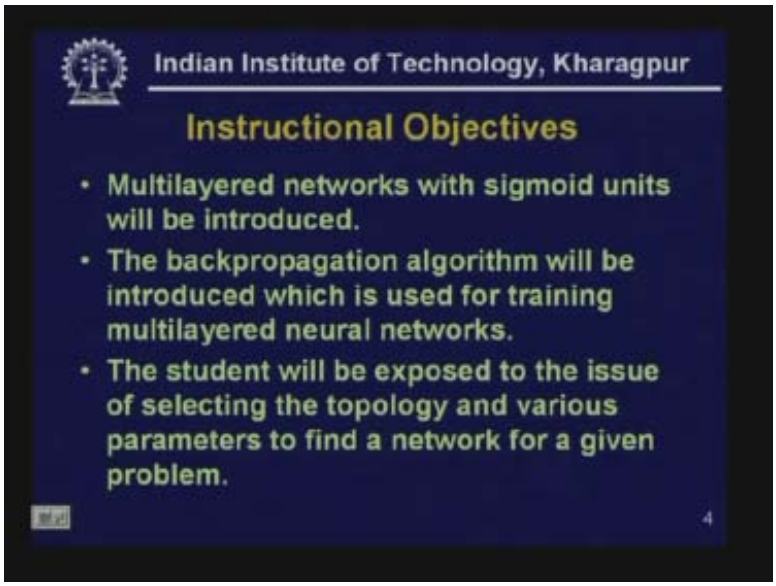


(Refer Slide Time: 1:16)



The objectives of today's lecture are as follows:
The student will learn what the limitations of single layer perceptron networks are. We will introduce multilayer networks. The student will become familiar with a sigmoid unit and the properties of neural network unit which uses a sigmoid function. The student will learn how gradient descent works with sigmoid units.

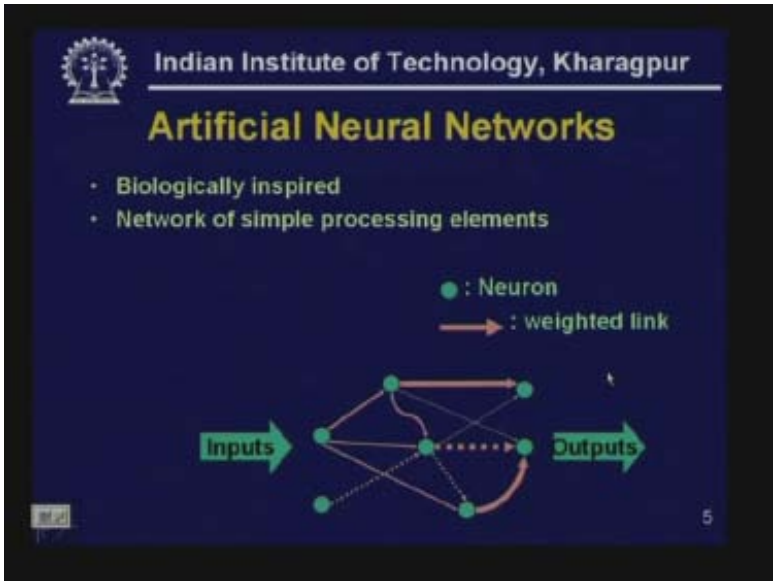
(Refer Slide Time: 1:52)



Multilayered neural network with sigmoid units will be introduced. The back propagation algorithm will be described which is used for training multilayered neural network. The student will also be exposed to several issues in designing a neural network. For example,

selecting the topology or architecture of neural network and tuning the various parameters so that the network is able to work on a given problem.

(Refer Slide Time: 3:07)

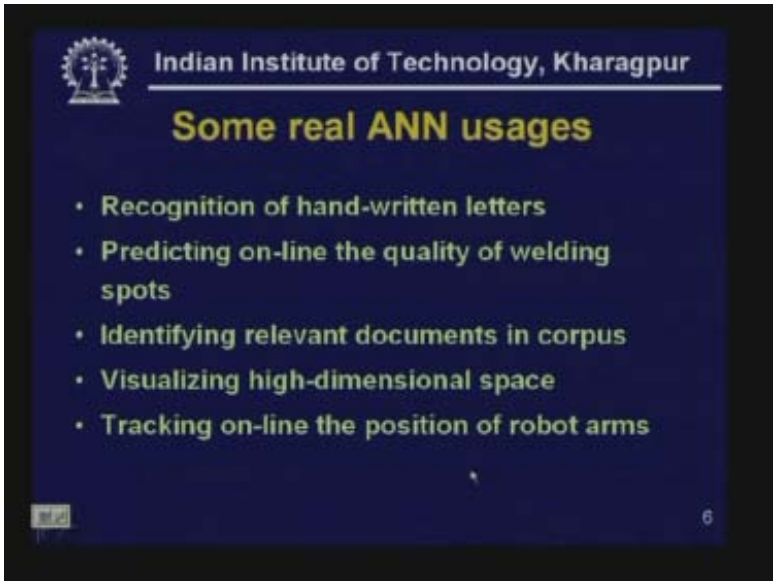


A neural network is a system which has been inspired biologically. We have a network of simple processing elements which are connected to each other via weighted links. The inputs are fed to the input units and as a result of the computation done in these units the outputs are produced.

Some uses or applications of artificial neural network ANN:
ANNs have been used for recognizing hand written letters, for predicting on-line the quality of welding spots, for identifying relevant documents within a corpus that is a large number of documents, the visualization of a high dimensional space and tracking on-line the position of robot arms.

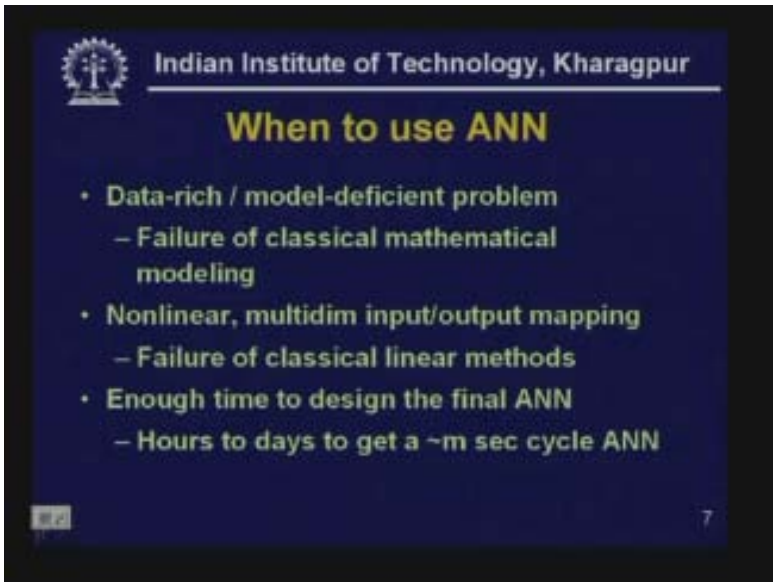
There are many uses that neural networks have been put into. These are a few of the utilities of neural network. When you are doing machine learning usually you will be given some problem. And you will have to find out what learning method is appropriate for tackling the problem. We have looked at a few learning methods namely the decision trees as well as now we are looking at neural networks. There are many other learning methods which have been used. Normally when you have a real life problem which is posed to you then you have to decide which learning method to apply. You have to guess which learning method will be most appropriate for a given type of problem.

(Refer Slide Time: 3:52)



Usually what are the types of problems for which the ANN can be considered an appropriate to reply?
Usually the problem is when we have got a lot of training data but we do not have a very good model of the data. Those are the problems for which we can apply artificial neural network. So problems that we cannot mathematically analyze or find a mathematical model to model the problems but we have a enough data then using neural networks is a good idea to model such problems. Neural networks also achieve nonlinear multidimensional input output mapping where the linear modeling does not work.

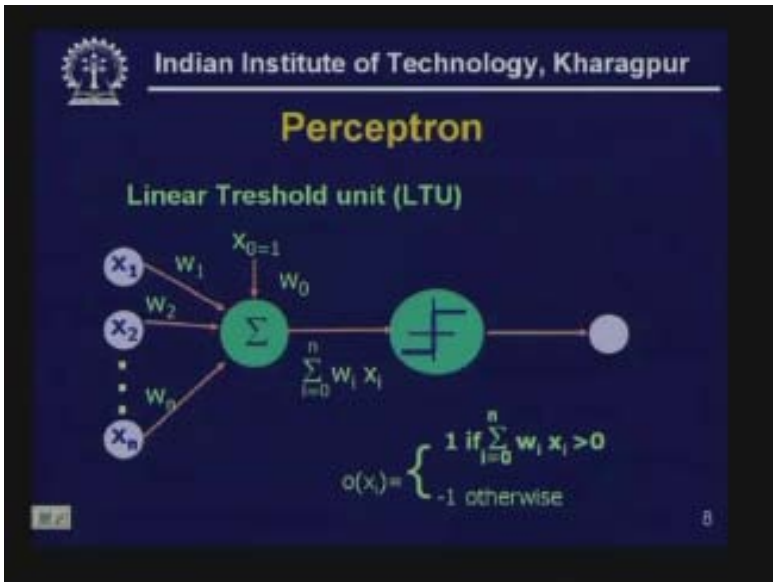
(Refer Slide Time: 6:22)



And also, if we are trying to model a problem with neural network we have to spend some time in finding appropriate network and training the neural network. So, even after we have decided to use a neural network it will take some time to tune the neural network topology as well as parameters as well as the training of the networks. So, some time will be spent doing that. Neural networks are also able to handle noisy training data very well.

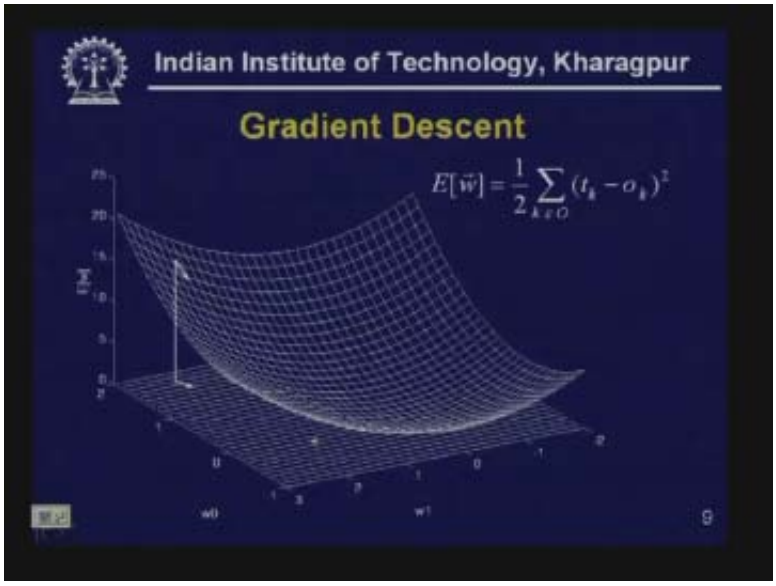
Decision trees are very good for data which is not always noisy, a lot of the data is nominal and when we require learn a function which humans can interpret and understand or rules that we can understand. In neural network when we get a hypothesis the hypothesis is represented by a set of weight values. So the rules are not symbolically or understandably present to the human user. This is the reason why neural network methods or often referred to as black box methods because we cannot interpret easily the rules that neural networks learn.

(Refer Slide Time: 7:45)



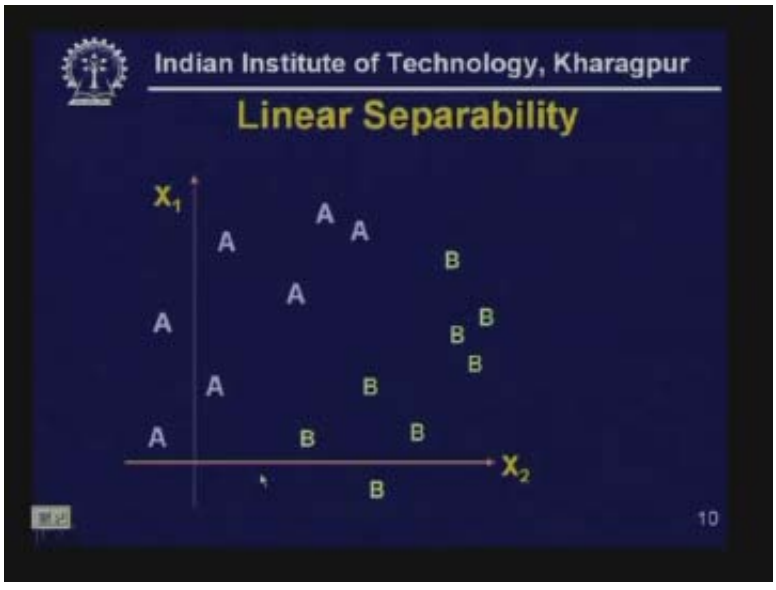
We looked at linear threshold units where we have a sigmoid unit which does a weighted summation of the inputs followed by application of a thresholding function or a step function.

(Refer Slide Time: 8:05)



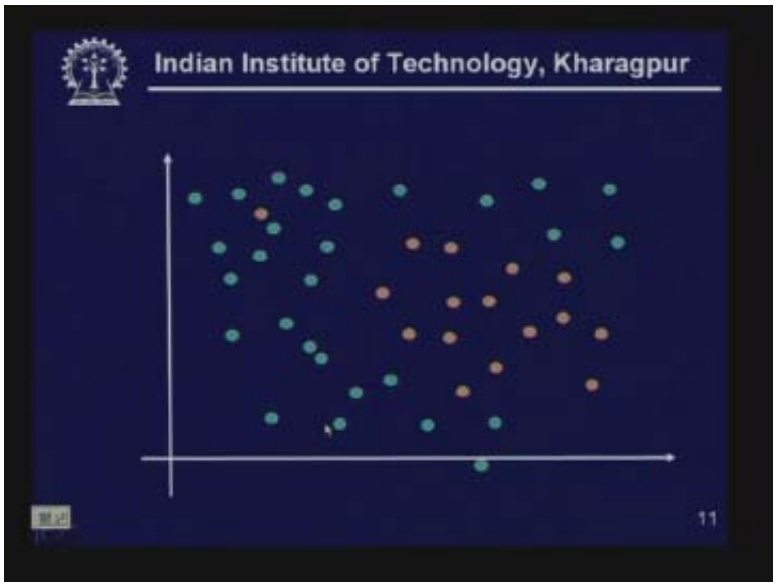
We also looked at, if we have a function which is continuous and differentiable how we can use Gradient descent to find out the set of weights for which the function has minimum error. So, if we have a thersholding unit which is a non differentiable function we will not be able to Gradient descent. But if we have simply the linear function, suppose we just have this summation unit and no thersholding unit in that case we can do Gradient descent on the weight surface and try to find the set of weights for which the error is minimum.

(Refer Slide Time: 9:00)



Linear units or even linear thersholding units are able to classify correctly those instances which are linearly separable. For example, here we have a set of points A and B. the A are the positive examples, B are the negative examples and we can find a line which separates the A points from the B points. Such problems can be tackled effectively by linear units or linear thresholding units.

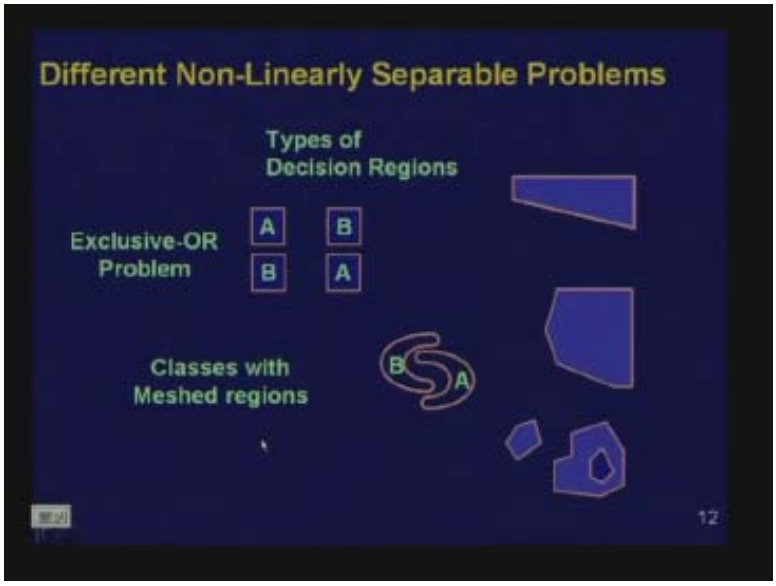
(Refer Slide Time: 9:48)



However, there are problems where the decision boundary is not linear where the linear decision boundary does not exist. In that case we will not be able to find a linear decision surface. Suppose we try to find a linear decision surface we might be able to find some surface and we will notice that there are some errors like we are trying to separate the green balls from the pink balls. On the right hand side there are some green balls. So we have not been able to achieve correct full separation of the positive points from the negative points.

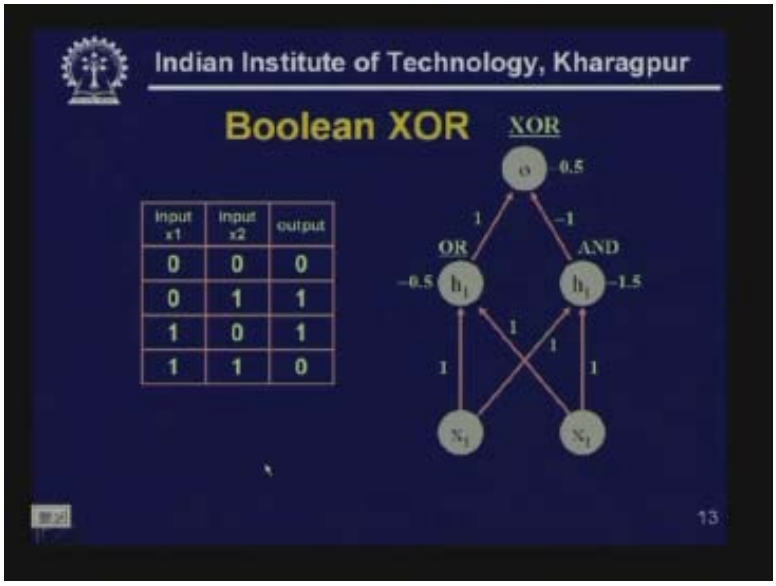
So what sort of decisions surface would we need in this case?
Suppose we had a decision surface which looked like this suppose we had this decision surface it can account for most of the pink points. It can separate most of the pink points from the green points even though there is one left over. So we can have the union of these two decision surfaces or we can only take this decision surface and ignore this smaller decision surface and treat it as noise. But in order to represent this decision surface a linear thresholding unit is not good enough.

(Refer Slide Time: 11:21)



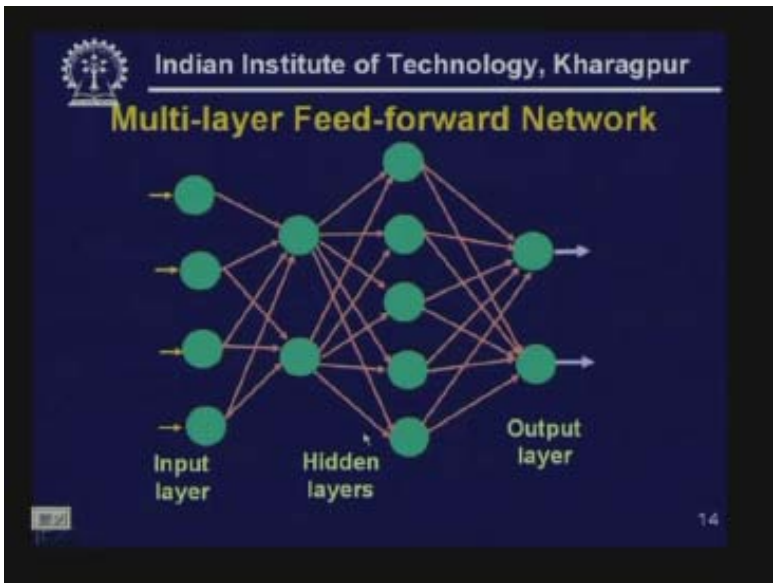
Decision regions: when we have a decision region where all the positive points are have this sort of shape that is all the positive points can be separated by a single line in those cases a linear perceptron does work. For problems like the exclusive OR we cannot have a good separation by a linear thersholding unit. There are problems when regions are meshed. And for such problems also we cannot use linear thresholding units. This is another example for problem. There we have two regions and with a hole in the region. So, only this portion is positive and the rest of the region the green region is all negative. So such problems cannot be separated by a linear unit and then problems where there is a positive region separated by negative region they also cannot be handled effectively by a linear thersholding unit.

(Refer Slide Time: 12:42)



The Boolean XOR function is one function which cannot be represented by a single linear unit. However, if we arrange the linear thersholding units in two layers, in this unit we compute x_1 and x_2 and here we can compute a XOR by setting these weights. So using these two layers of units we can compute the XOR function. So if we add a layer to the network that gives us computing power and which lets us go beyond linear decision surfaces.

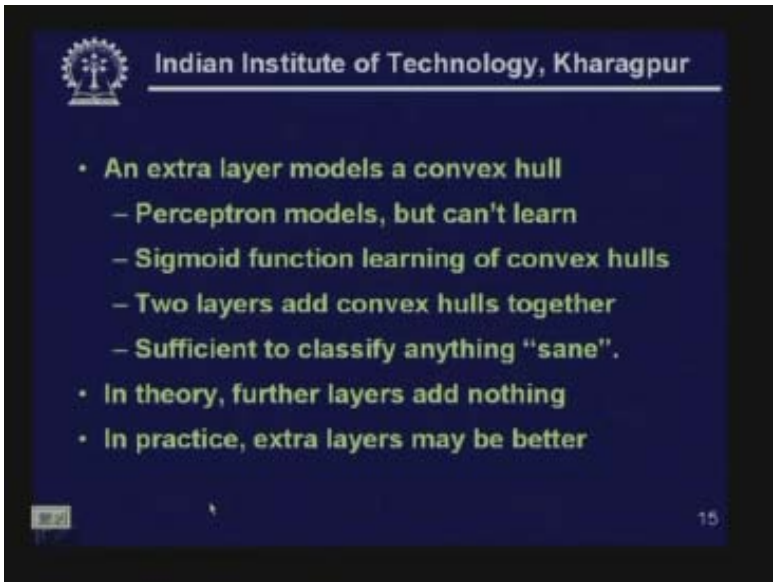
(Refer Slide Time: 13:59)



Multi-layer feed forward network: In multi layer feed forward network we have an input layer where the inputs are feed in, we have an output layer where the outputs go and

within we have hidden layers. These are called the hidden layers which are not visible at the output and where the inputs also do not fit in. The hidden layers are the ones where this no input output but they help in computing the function at the output.

(Refer Slide Time: 14:40)

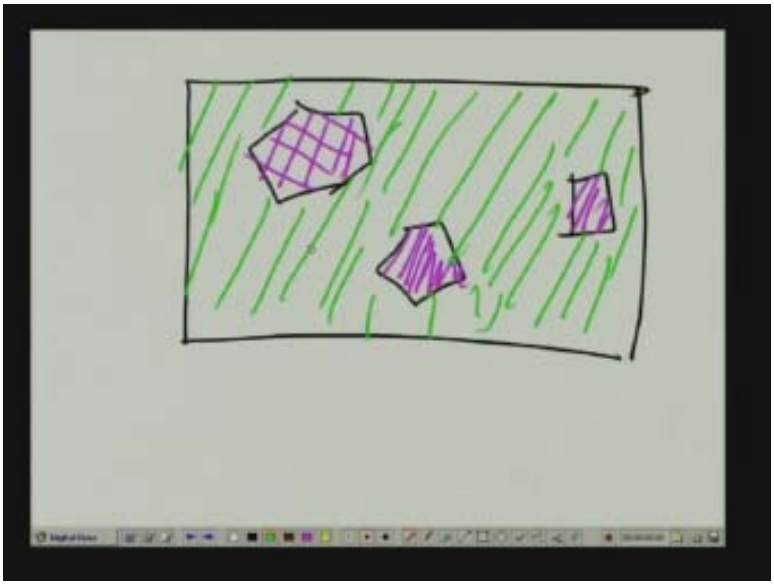


So, if we had a hidden layer what can we represent? If we had one hidden layer that is if we have a two layered neural network such a network can represent any Boolean function. Any Boolean function can be represented by a two layered network. Any continuous differentiable function can be represented by a three layered network. That is the network with one output unit and two hidden units. Geometrically if we have one hidden layer that is a two layered neural network having one hidden layer we are able to separate positive regions that come within a convex hull.

Suppose the green region is the negative region the pink region is the positive region and the positive region can be enclosed by a convex hull such a separation can be learnt if we have one hidden layer. if we use a two layered neural network comprising a perceptron they can model such learning problems. However, perceptrons are not easy to learn using gradient descent.

When we use a thersholding function because the thersholding function is non-differentiable we cannot use gradient descent if you are using perceptrons. Today we will look at the sigmoid function which is differentiable and using which we can learn functions that can be represented by convex hulls. If we have a two layered network then we can learn a collection of convex hulls. For example, suppose we have a problem where let us say the positive region is mapped by a number of convex hulls, so this is the positive region given in pink and the rest of the region is negative region given in green. Therefore such a decision surface can be learnt by a neural network having two hidden units. In the first hidden unit you can learn the individual convex hulls and in the second unit we can combine them.

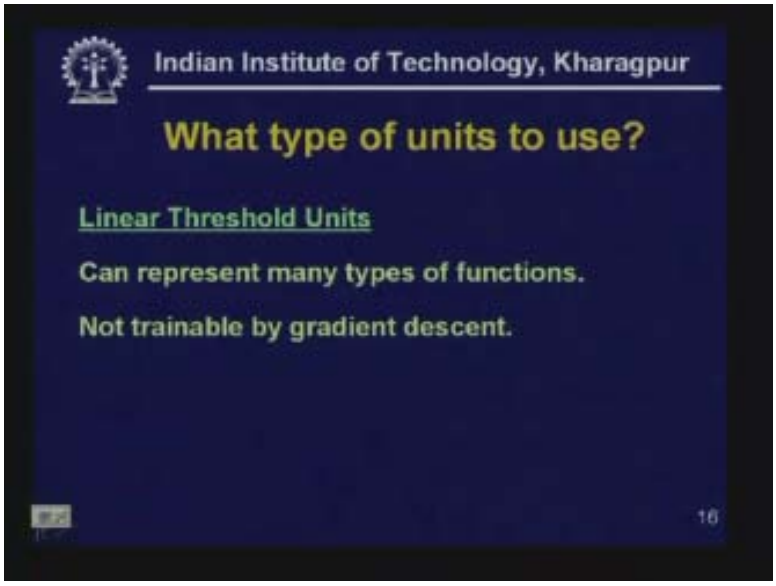
(Refer Slide Time: 17:41)



Therefore when we use a two layered neural network most problems can be represented. In theory if we add more than two hidden layers it does not gave us any representational power. But merely the fact that the two layered network or a three layered network has a certain expressivity it does not mean that the learning problem is simple.

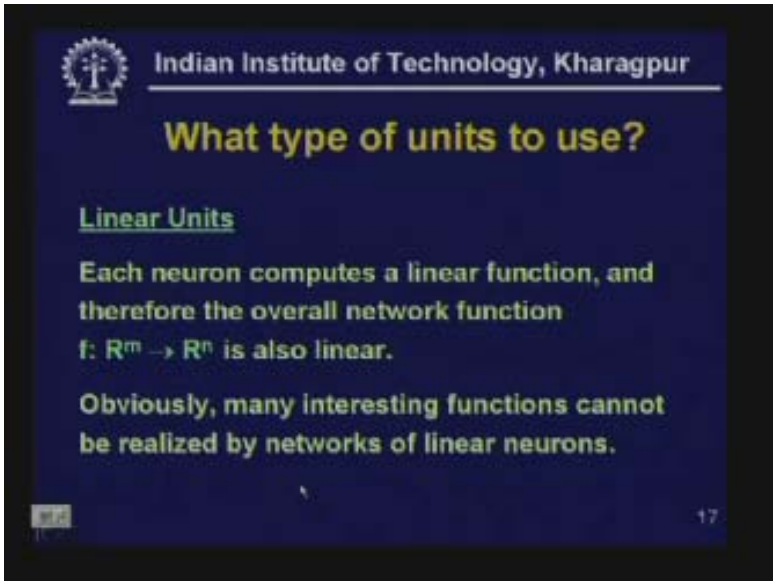
Given a learning problem one has to find out how many layers are needed for solving the problem and for each layer how many hidden units are needed, after that the training has to take place. Therefore when we go for multilayer neural network the training does not guarantee that we are able to learn the optimum network. So, the problem remains hard but a lot of problems have been solved successfully by neural networks.

(Refer Slide Time: 19:11)



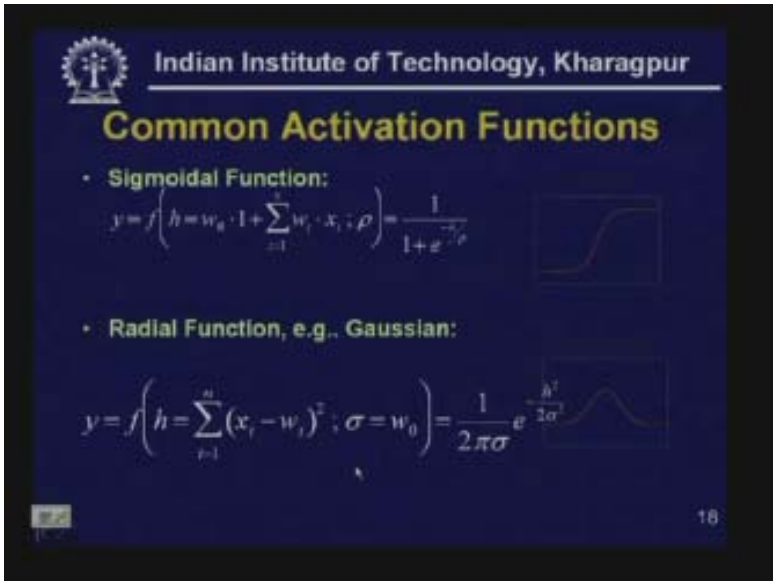
We discussed linear threshold unit and said that they can represent many types of functions but they are not trainable by gradient descend. We looked at linear units we charge differentiable. Unfortunately when we want to go for multilayer neural networks and if we take several linear units together we do not add to representational power.

(Refer Slide Time: 19:51)



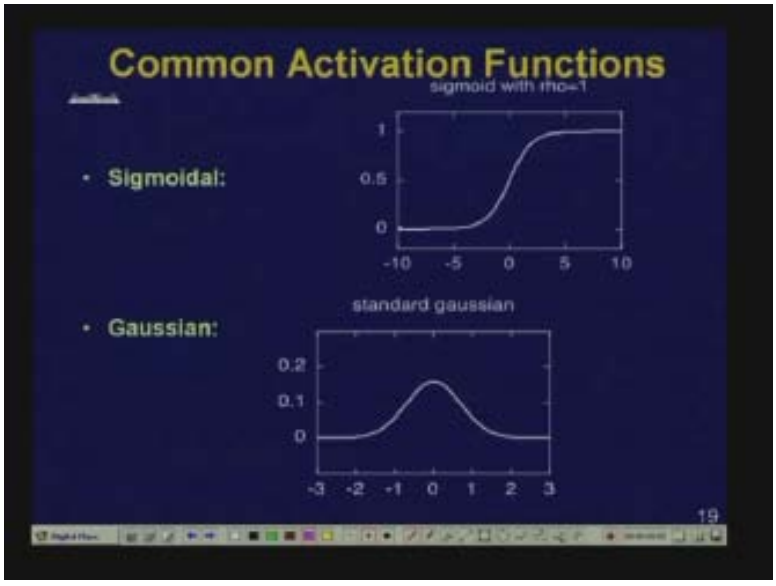
So, if we add several layers of linear units what we get is in effect another linear unit. Therefore it does not have the power to represent all types of decision surfaces. So, the type of functions we can represent by multilayer neural networks in a linear unit is [……] 19:55. So there are other activation functions that we can consider.

(Refer Slide Time: 20:51)



We do not want to use linear threshold unit because it is not differentiable so we will try to use functions which are continuous and differentiable and at the same time which give us representational power. Hence the two such functions are; the sigmoidal function represented as y is equal to 1 by 1 plus e power minus h by p and the radial function or the Gaussian function which is given by y is equal to 1 by 2 by sigma e power minus h square by 2 sigma square.

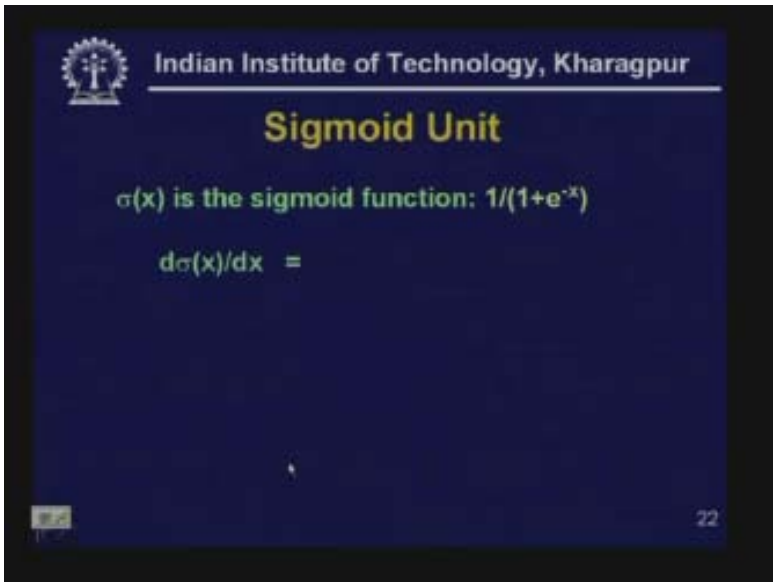
(Refer Slide Time: 21:03)



So the sigmoidal function and the radial function are examples of two functions which people have used in multi layer neural network. Sigmoidal function: The sigmoidal

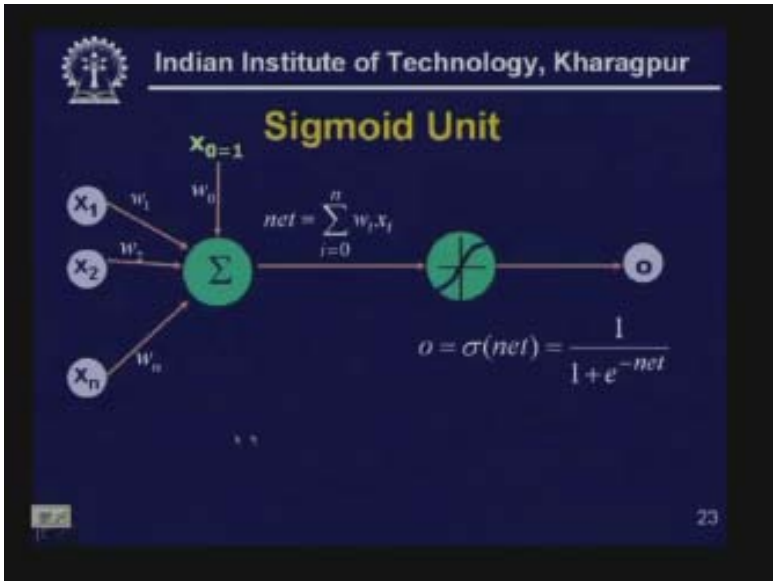
function has a shape which looks like this: So, as we can se it is an S shaped function but it very closely resembles the step function with the exception that it is continuous and differentiable. This is very similar to the state function. The Gaussian function on the other hand gives us a bell shaped curve which is also used in representing neural network. So this is the sigmoidal function again; y is equal to 1 by 1 plus e power x.

(Refer Slide Time: 21:48)



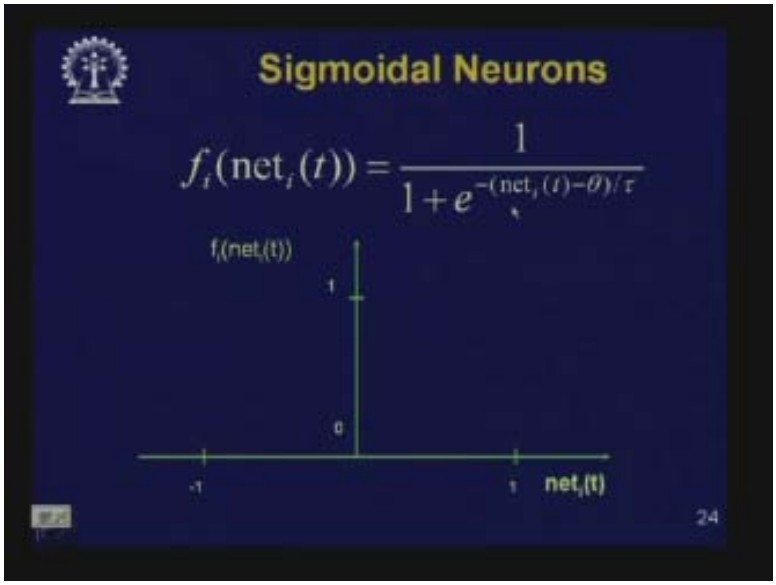
The sigma x is the sigmoidal function and if we differentiate sigma x with respect to x we get, you can do the differentiation, it is an algebraic manipulation and you will get that ddx(sigma x) which is nothing but sigma x into 1 minus sigma x.

(Refer Slide Time: 22:15)



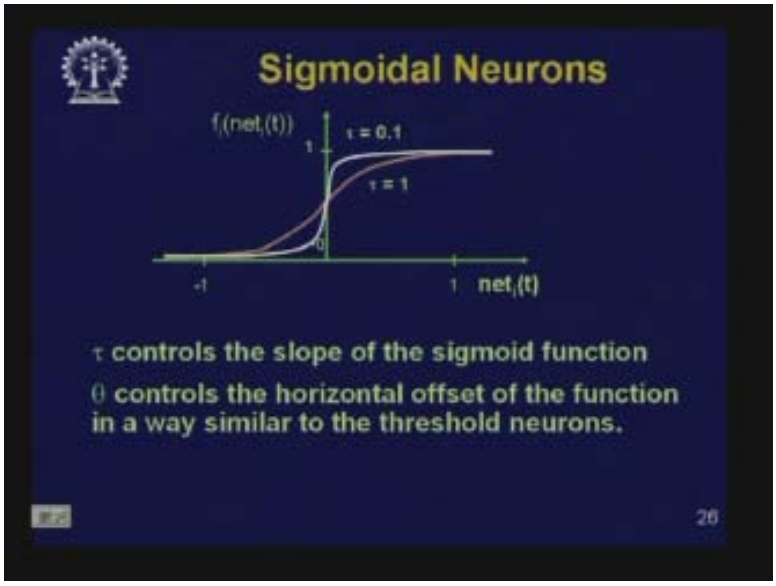
The mathematics becomes much simpler when we deal with sigmoid function because not only it is differentiable but we can express it in the form of a function itself. So the sort of unit we use for a neural network is called the sigmoid unit as follows: There is a summation unit followed by the application of the sigmoid unit. The summation unit computes sigma $w_i x_i$ and the sigmoid unit computes sigma of this quantity. Hence if net is sigma $w_i x_i$ the output is sigma of net which is 1 by 1 plus e power minus net that is 1 by 1 plus e minus sigma $w_i x_i$.

(Refer Slide Time: 23:28)



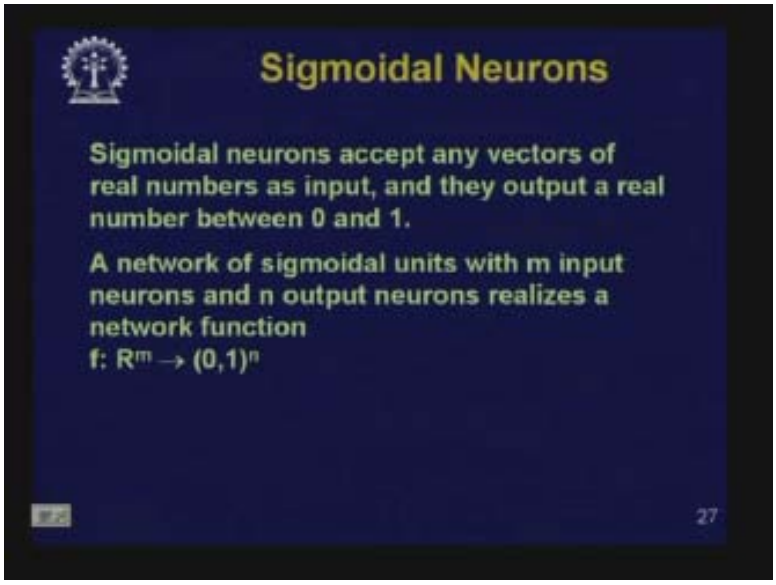
Now a more general form of sigmoid function is this; 1 by 1 plus e power minus net minus theta by tau. Here theta is the threshold. If theta is equal to 0 the S shaped function we get is placed around origin. If we put theta as some other value we will be able to give a shift to this function. And tau 1 by 1 plus e power minus net by tau where tau is a sort of stiffness cost and varying the value of tau we can vary the slope of function that we get. For example, if tau is equal to 1 this pink curve shows us the resulting sigmoid function. If tau is equal to 0.1 then we get this white curve which is steeper than the curve with is equal to 1. So, if I make tau less than 1 we get a function which more closely resembles the step function. So tau controls the slope of the sigmoid function.

(Refer Slide Time: 24:42)



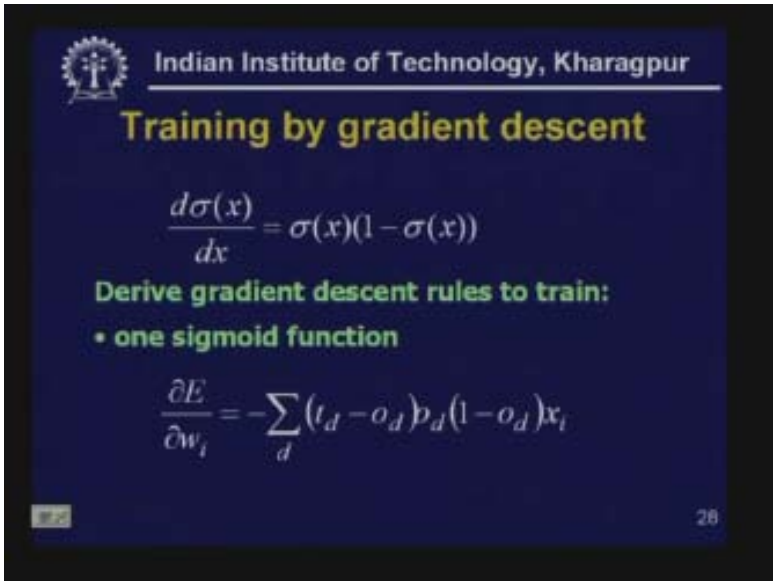
Smaller the value of tau higher the slope, theta controls the horizontal offset of the function in a way is similar to threshold neurons.

(Refer Slide Time: 24:42)



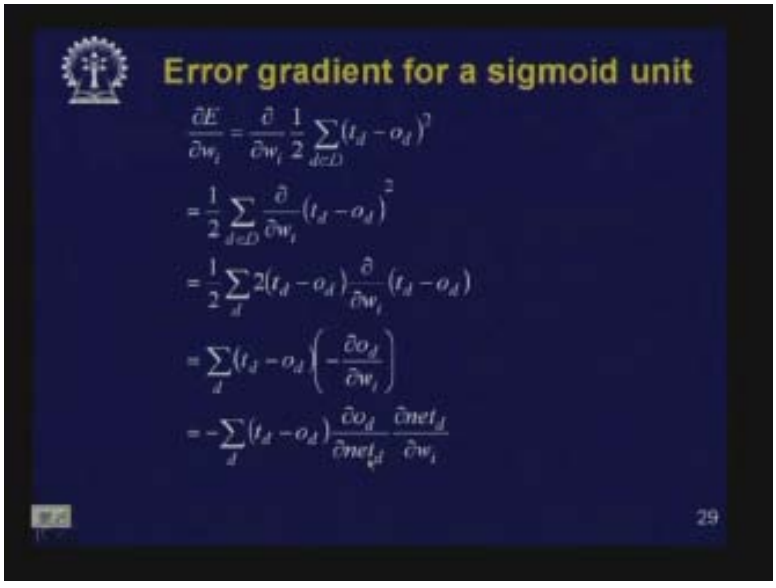
Sigmoidal neurons can accept any vector of real numbers as input and they output a real number between 0 and 1. A network of sigmoidal units with m input neurons and n output neurons realizes a function that maps r power m minus 0 1 power n.

(Refer Slide Time: 25:21)



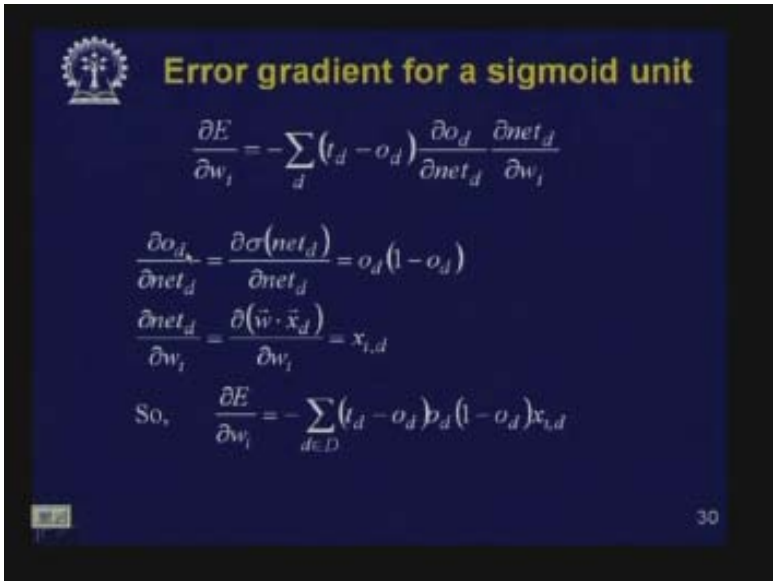
Now let us see how we can train using gradient descent. We have seen that d sigma x dx is equal to sigma x into 1 minus sigma x. Now we will try to derive the gradient descent rule when we have a single layer of sigmoidal units. Now what we will try to do is we will define the error function and try to find out the partial derivative of the error function with respect to each value of w_i and we will try to compute those values. And we will see that value is equal to minussigma over all training instances d t_d minus o_d into o_d into 1 minus o_d into x_i. Here t_d is the target value of the d^{th} training example and o_d is the actual output that we get using the neural network and xi is the i^{th} input to which the corresponding weight is w_i.

(Refer Slide Time: 26:43)



Now here is the derivation for this:
Del e by del w_i is the partial derivative of the error function with respect to w_i is del del w_i and e is written as 1 by 2 (t_d minus o_d) whole square sigma summation(t_d minus o_d) whole square. So this is the half of the sum of the square error this is the definition of error function. Now, if we differentiate it with respect to w_I, first of all we can do some arithmetic manipulation we can bring half outside the sigma and we get 1 by 2 sigma del del w_i (t_d minus o_d) whole square. Then differentiating this we get two into t_d minus o_d times del del w_i t_d minus o_d by change of a variable we get this. Then what is del del w_i(t_d minus o_d)? It is minus del o_d by del w_i because t_d does not depend on the wi. But o_d depends on w_i because o_d is obtained by doing this summation and then the sigmoid. So we have sigma summation of o_d t_d minus o_d into del o_d by del w_i. Now del o_d by del w_i we can write using the chain rule as del o_d by del net_d times del net_d by del w_i. And we can simplify this further; del o_d by del net_d.

(Refer Slide Time: 28:38)

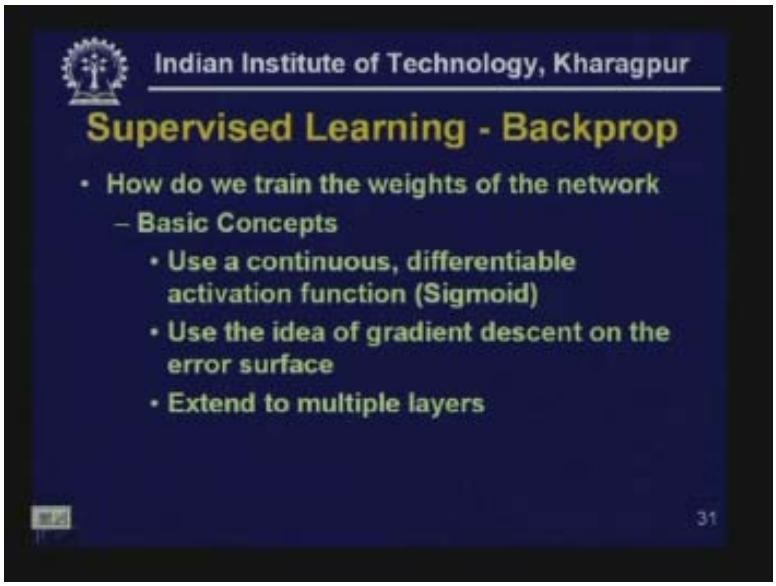


What is o_d?
o_d is nothing but sigma net_d. So del(sigma net_d by del net_d) is nothing but o_d times 1 minus o_d because this is a sigmoid unit. And what is del net_d by del w_i? net_d is nothing but the dot product of w and x sigma $w_i x_i$ and net_d is w. x_d del w_i. Now you see that in the w. x_d we have $w_0 x_0$, $w_1 x_1$, $w_2 x_2$ but now none of these terms depend on w_i except the term from $w_i x_i$. So this is nothing but x_i for the d^{th} training example. Therefore ultimately what we get is that the partial derivatives of the error function with respect to the weight w_i is minus of sigma over all training examples t_d minus o_d into o_d into 1 minus o_d into x_i d.

So this is the slope with respect to a particular weight w_i. We can find with partial derivative with respect to all the weights and we can find the components in all directions of this slope so we can compute the slope. Once we compute the slope in gradient descent what we do is we find the direction opposite the slope. We want to climb down so we

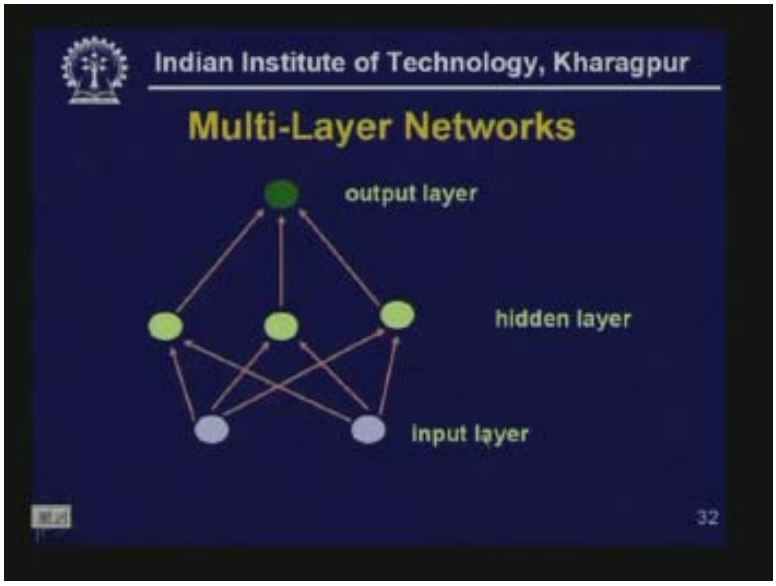
find the negative of this slope and we take a step in that direction. Now let us see how to train the weights of network. The basic idea is that we will use continuous differentiable activation function which is represented by a sigmoid unit.

(Refer Slide Time: 30:39)



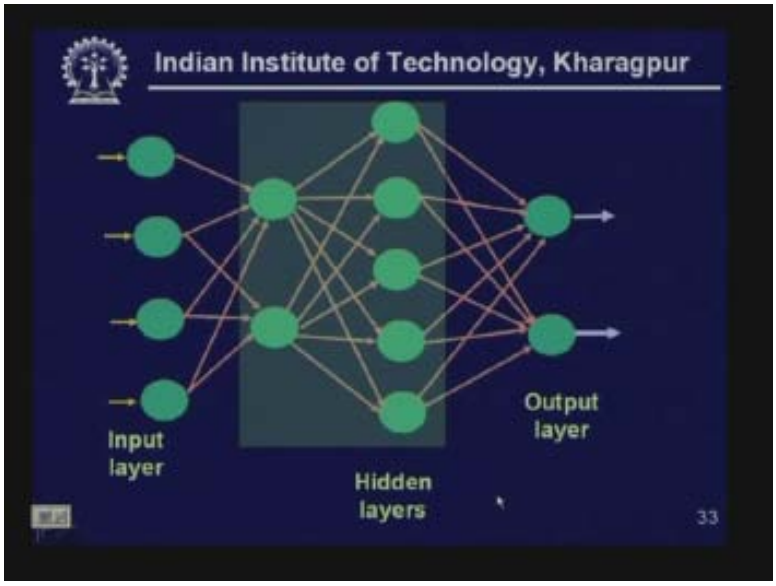
We will use the idea of gradient descent on the error surface and we will try to extend this to multiple layers.

(Refer Slide Time: 30:49)



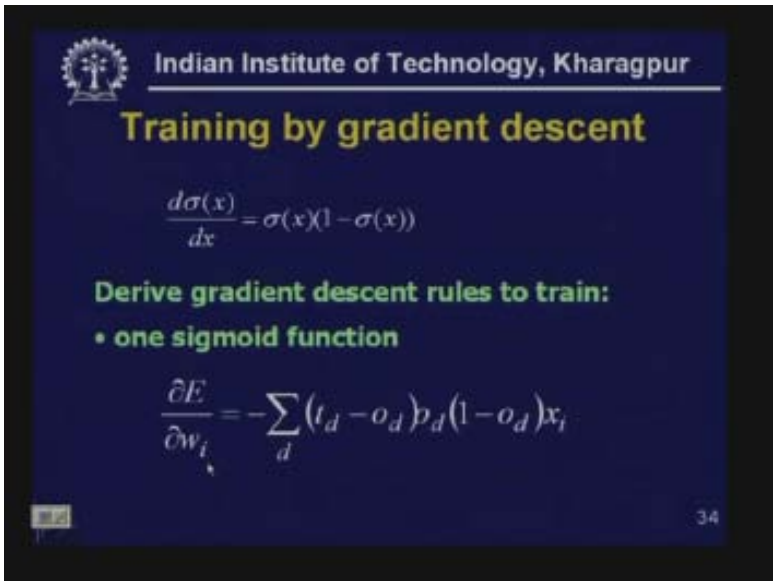
So this is a schematic diagram of multilayer network with an output layer, input layer and hidden layer. Here also we have two hidden layers.

(Refer Slide Time: 31:05)



We have been able to derive the gradient descent rule in one sigmoid function and we have seen that del e by del w_i is minus of sigma by d t_d minus o_d into o_d into 1 minus o_d into x_i.

(Refer Slide Time: 31:15)



Now, what if we have multiple layers of sigmoid units? If we have multiple layers we will use a technique which we call back propagation. Now, we will give the back propagation algorithm but before we do that we will try to give you simple idea of what is back propagation and how it works. So the basic idea is that, suppose you have some input units here and you have some outputs here and then you have some hidden layers

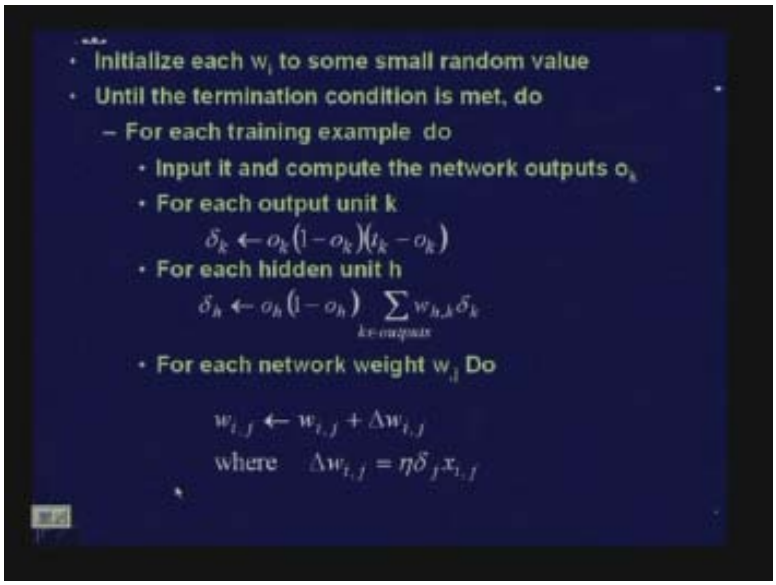
here. Now, let us look at the output unit. When do you change the weights on the different arcs that lead to the output? We only change the weights if the output we get from the network does not agree with the target output. So, at the output unit we are able to recognize if there is some error.

Error is recognizable at the output unit and we can try to change the weights of the different arcs that lead to the output unit. We change the weights so that this difference between the target value and the output value is minimized. But if you have two layered neural network how we know that there is an error at the hidden unit. And how do we update the weight values because we do not know the target value at the hidden unit. At the output we know what the target value is.

At a hidden unit we do not know what the target value is. And if we do not know what is the target value we will not know what the error is and we will have no basis for modifying the weights. So the basic idea behind back propagation here is that whatever error you observe at the output you try to allocate the error to the hidden units. If there is no error at the output that means both the outputs have no error you assume that there is no error at the hidden units also. But if there is some error here you try to allocate the error back to the hidden units from which it receives the input.

Similarly, if you have an error at this output unit you allocate this error to the units from which it receives input. So we propagate the error from the output backwards to the hidden units. So how we allocate the error? We allocate the error in proportion to the weights. So, once we allocate the error we know the target value that we have to minimize and therefore we are doing gradient descent. So this is the basic idea of back propagation.

(Refer Slide Time: 31:15)



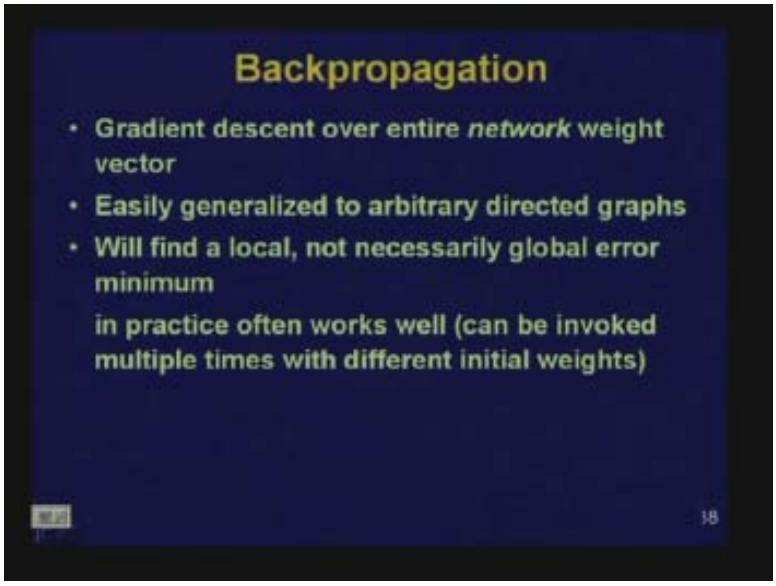
Back propagation algorithm:

In the first step we initialize the weights to some small random values. We first decide a topology of neural networks which has some arcs with each arcs there are some weights, we give them small randomly selected values. After that we feed the training example to the network. We take each training example and input it to the network and we compute the network outputs ok for each output unit k. And then we will do gradient descent with back propagation.

For each output unit k we compute delta k to be ok into 1 minus ok into t k minus ok. For each hidden unit h we compute delta h to be oh times 1 minus oh times sigma w_h delta k. Therefore each hidden unit takes the burden of some of the error at each of the output units to which it is connected. So delta k is what is coming from the k^{th} output unit and $w_h k$ is the weight of the arc connecting this hidden unit with the k^{th} output unit. So we compute delta k initially at the output nodes then at the node above the output node then at the node above that and so on.

So, after we have computed delta k at all the units such as the output units as well as the hidden units then we update the network weight as follows: w_{ij} is updated as w_{ij} plus delta w_{ij}. and delta w_{ij} is equal to eeta delta j times x_{ij}. Therefore the actual algorithm is quite simple to implement. so there is an initialization phase, after that there is a feeding phase where the training examples are fed to the neural networks and then at each unit starting from the output up towards the input we compute the value of delta i and then we modify the network weights and then we continue if the network is not satisfactory. So, in back propagation we do gradient descent over the entire network weight vector.

(Refer Slide Time: 38:45)

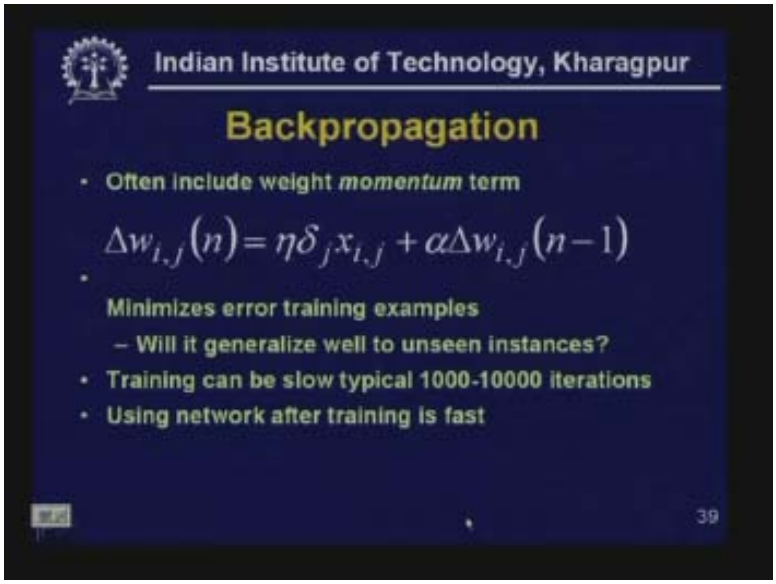


This back propagation can work not only for a neural network with one hidden layer but neural network with any number of hidden layers. In fact it can work even though we do not have layered neural networks but neural networks in the form of acyclic directed graph. We can still do this back propagation algorithm. The basic idea is that we start

from the output and then find the delta value at those notes and then we find the delta value of those nodes for which downstream of the entire delta values have been computed.

Therefore we compute the delta values backwards and then we can do the weight training. So, back propagation is not an optimum algorithm. By doing back propagation you cannot guarantee that the weight vector that you arrive at is the best weight vector. But in practice back propagation often works well. In fact what you can do is if you run back propagation once with some initial input values and you get a network which does not satisfy you completely you can still run back from back propagation several times with different initial weight values. There are some variations to back propagation.

(Refer Slide Time: 40:23)



Apart from the delta term that is eeta delta x_{ij} some people add another term called the momentum term.

What is the momentum term?
Momentum term takes contribution from the previous value of delta w_{ij}.

Why is the momentum term used?
Sometimes what happens is because there is a local minimum and as a result of back propagation the system may have a tendency to get stuck in the local minimum. So this is the local minimum and there is a scope of going further down in the error surface. So, to prevent your current weight vector to get stuck at the local minimum what we do is we keep track of what is the previous direction in which the error surface is moving.

Even if we get stuck in the local minimum we try to have the momentum of the fall so to the new delta w_{ij} value we add the value of the previous slope so that our weight function

can escape from the local minimum. So, by using the momentum term we can minimize the error in training examples.

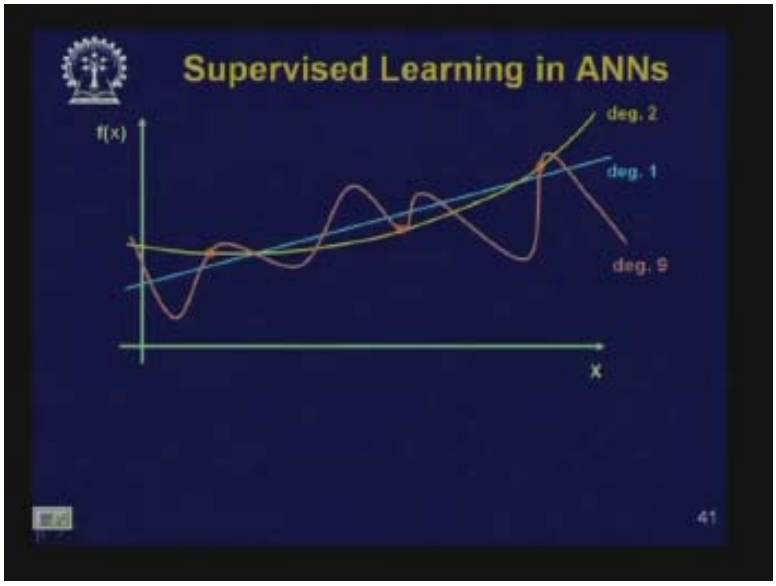
Now what we will do is we will study several issues concerning neural networks. For example, we wish to know that suppose we have trained a neural network using our training example how to find out whether works very well for unseen training examples. Secondly, one thing we have to keep in mind when we use neural networks is that usually neural network training time can be quite high because we need several thousand iterations for training to converge. However, if we are able to train a neural network using the neural network is very fast. The feed forward nature using the neural network is very fast. Normally when we use a training example which we use for learning as training time increases the error reduces as shown by this graph.

(Refer Slide Time: 40:23)

With time as we train more and more typically the error reduces. However, as we noted when we looked at decision trees if we look at new examples which we do not use for training then with them the error curve may be different. For example, with the new example usually error reduces some time and then the error starts increasing and this is due to over-fitting.

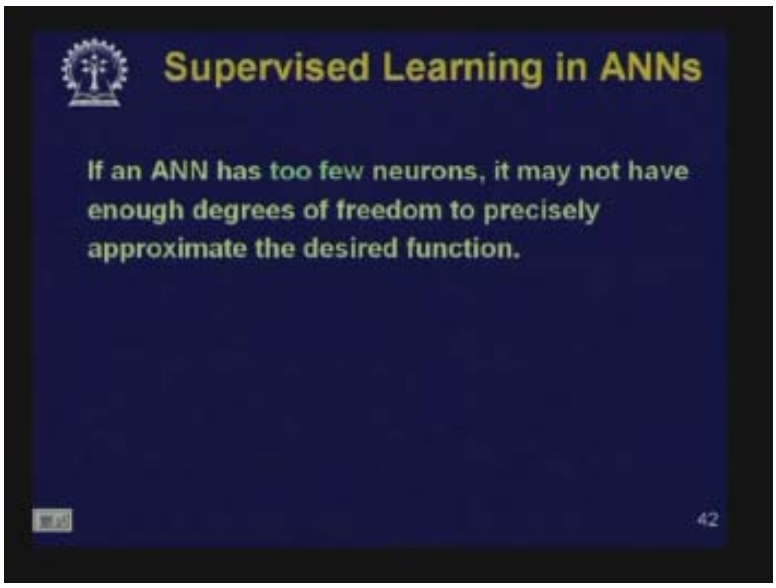
We would like to detect when the error is at minimum and we would like to stop at the point where the error on the test set is minimum. Now how to detect this point? As we have done earlier what we can do is we can have a separate validation set or test set to decide when to stop training the network. Now suppose we have three points; if we have a very simple function connecting these three points often there is a greater chance that such a function can fit unknown examples better.

(Refer Slide Time: 44:56)



If you have more complex functions as fitting the points such a function maybe over-fitting the data. Similarly, when we consider artificial neural networks sometimes we have to look for the simplest neural networks that can reasonably fit the data rather than a very complex topology which is able to fit the data perfectly. So, in a neural network when we have too few neurons that is if you have too few hidden units the network may not have enough degrees of freedom to precisely approximate the desired function.

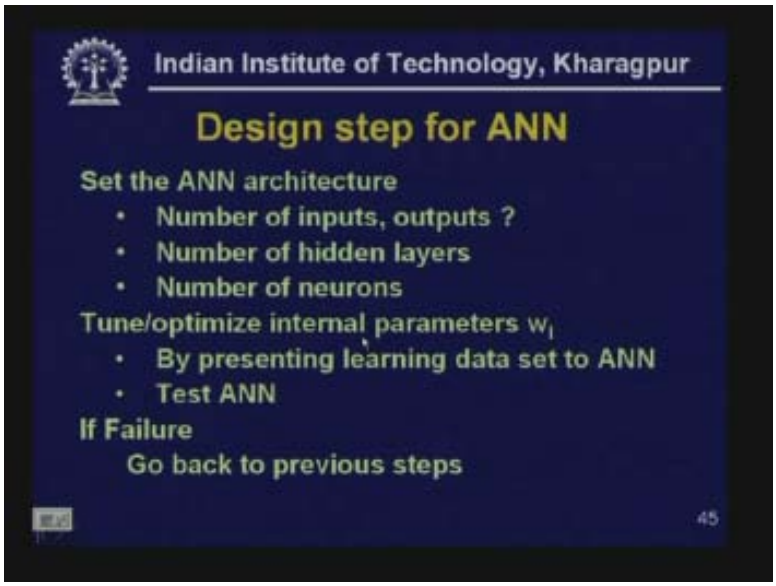
(Refer Slide Time: 45:46)



But if the network has too many neurons it will learn the training examples perfectly. But due to these additional degrees of freedom it may be over-fitting the data. So it may be

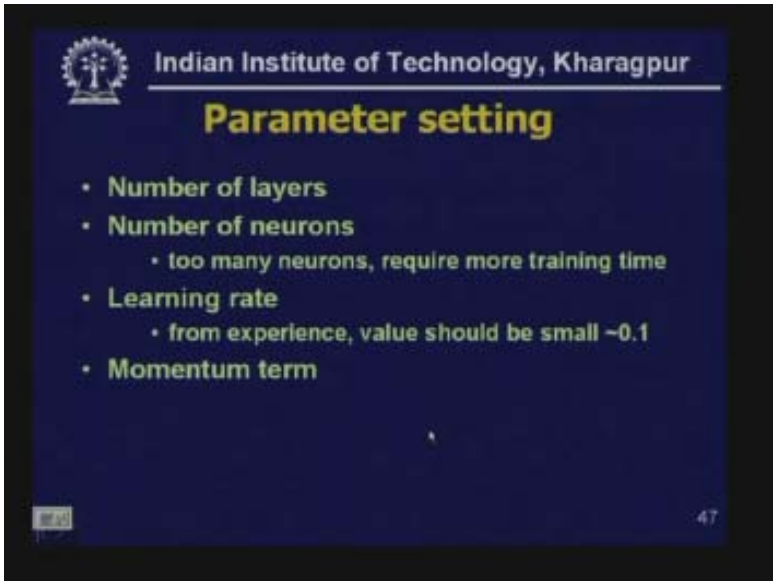
showing impossible behavior for unknown inputs. So when we design a neural network we have to be careful about that. Some other problems of neural network is that there are many parameters to be set. Suppose we are using the sigmoid function we have to decide the threshold of the sigmoid unit function, we have to decide eeta which is the learning rate, we have to decide alpha, the term associated with the momentum, we have to decide number of hidden units, we have to decide number of hidden layers and so on. Therefore, in a neural network using a neural network takes a lot of time as you have to experiment with all these and try to find a good network to fit your data. And coupled with this is the long training time. For each configuration that you have to take you have to train it and training time is typically quite long.

(Refer Slide Time: 47:13)



What are the design steps for an artificial neural network?
First you have to set the architecture for the neural network that is you have the inputs and number of outputs, you have to decide which inputs to take how many outputs to take, number of hidden layer that you have, the number of neurons that you will have in each hidden layer then you have to run your gradient descent algorithms to optimize the weight vector values and finally you test the network and if the success and the network is satisfactory then you are done but if the network is not satisfactory then you have to go back to the previous steps so that you try with different values of parameters and may be even different topology.

(Refer Slide Time: 48:00)

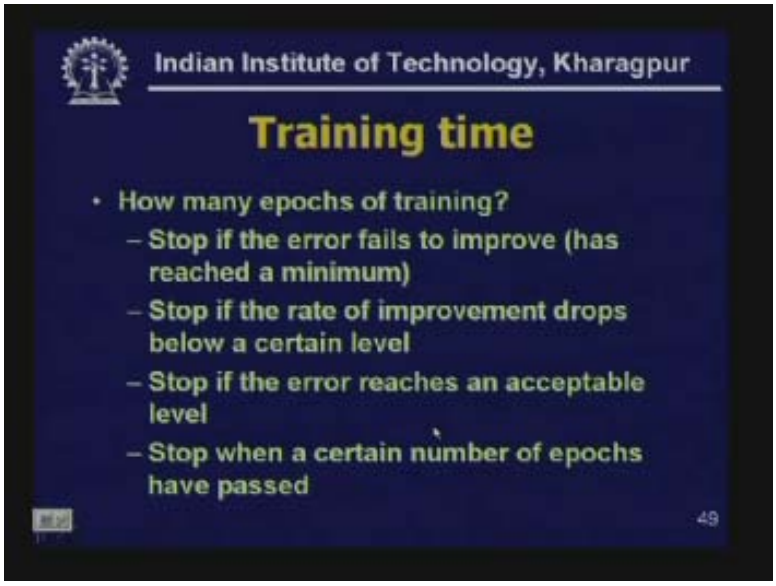


There are a lot of parameters we have to decide when we are working with the neural network.

(Refer Slide Time: 48:03)

And as we said over-fitting can usually occur. When do you terminate training? So normally you stop if the error fails to improve or you stop if the rate of improvement stops below a certain level.

(Refer Slide Time: 48:16)



Or you stop if the error reaches an acceptable level or you stop when a certain number of epochs have passed.

(Refer Slide Time: 48:37)