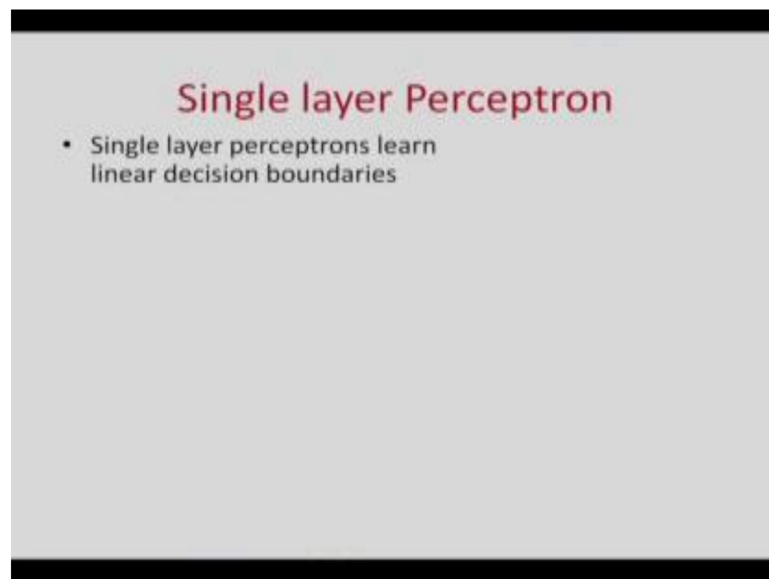
Introduction to Machine Learning
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module - 6
Lecture - 29
Neural Network and Backpropagation Algorithm

Good morning, today we will talk about Multi layer Neural Network and the Backpropagation algorithm.

(Refer Slide Time: 00:29)



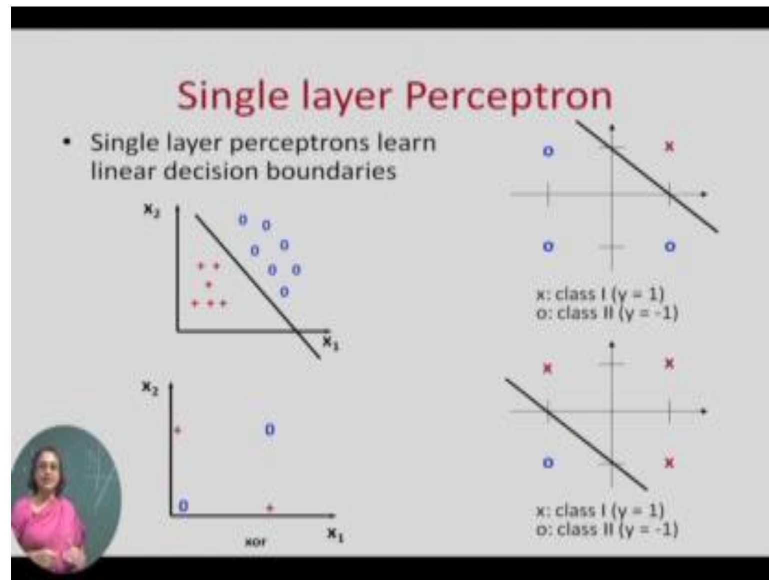
To refresh your memory, let us see, what we can do by a single layer neural network.

(Refer Slide Time: 00:40)

If we use a single layer perceptron, which we have already seen in a single layer perceptron, we have linear summation of the input units followed by non-linear function and the non-linear function, in the case of perceptron is a thresh-holding function, but we could also use other function such as the Sigmoid function, the Tanh function or the Relu function. So, we do sigma of sigma w i x i if you are using the sigmoid function or in general we can use function phi.

Now, if you have a single layer neural network or single layer perceptron it can the decision boundary is represented by a straight line. Suppose, x 1 and x 2 are the 2 features that you have the decision boundary is a straight line and such units will work to represent functions where the 2 classes examples belonging to 2 classes can be separated by a straight line, for example, if you look at the slide.

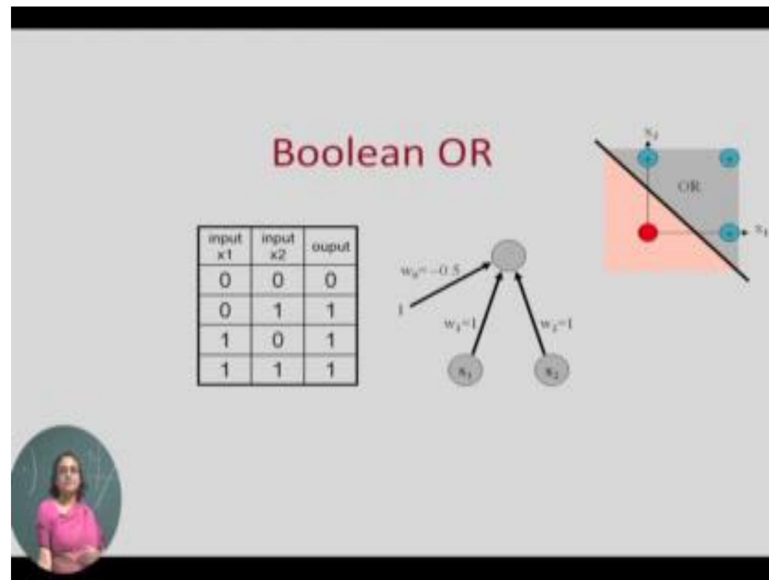
(Refer Slide Time: 01:50)



Here, we have 2 classes; 1 class 1 denoted by red class, class 2 denoted by blue circle and there is a straight line function that separates them. On the right, we see 2 other examples, where we have class 1, which is red; class two, which is blue and there is a line separating class 1 from class 2 and there exist linear decision boundary which separates class 1 from class 2.

Now, let us look at this fourth example. In the bottom left here, we have 2 red points, 2 red plus points 2, blue 0 points. Now, these examples we cannot have a linear boundary which separates the blue points from the red points. So, such machine learning problem cannot be represented by single layer perceptron.

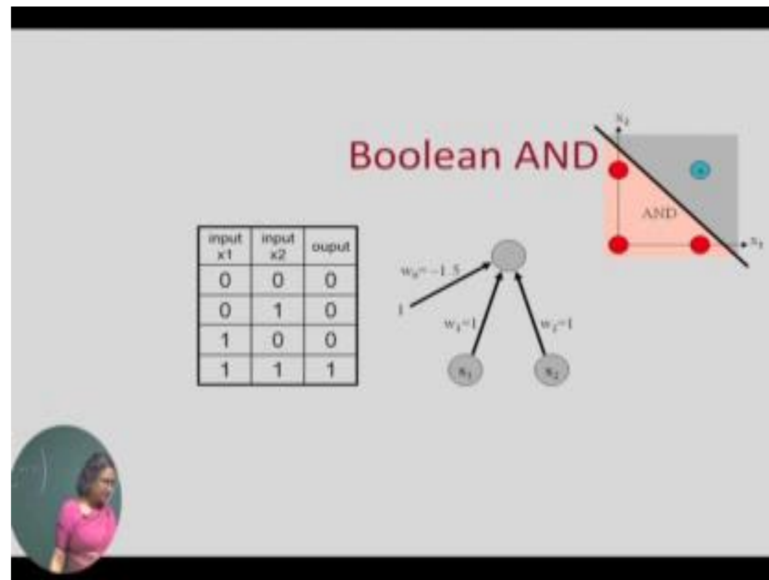
(Refer Slide Time: 02:48)



Now, this is an example of a function which can be represented by a single layer perceptron and this particular function is the Boolean OR function which most of you are familiar with the Boolean OR function outputs one, if any of the inputs is 1 and output 0, only if all the inputs are not 1 now this function can be represented by a single layer perceptron and learning the function means learning the weights on the corresponding edges.

So, there are 3 weights associated with this perceptron, w 1 from x 1 to the unit w 2 from x 2 that the unit and w 0, which is the bias now this particular diagram shows you that some specific values of w 0, w 1 and w 2 for which we can implement this function this function can be implemented by some sets of values this is an example of a set of values which can implement this function. So, this is the representation of the OR function and given this particular values of w 0, w 1 and w 2, we get a linear decision boundary which separates the space into 2 regions. So, that the plus points are in 1 region the minus points are in another region.

(Refer Slide Time: 04:20)



Similarly, this is another example of a function which can be represented by a single layer perceptron. This is the Boolean AND function, where the output is 1 only if both the inputs are 1 otherwise the output is 0, it is represented by this diagram and thus decision layer corresponding to some specific weights right. So, for example, if I said w 1 equal to 1 w 2 equal to 1 and w 0 equal to minus 1.5, it is it will denote decision surface which works for this and function there could be other combinations of weights also which work for the and function, but this is 1 combination of weights which implements the and function.

(Refer Slide Time: 05:10)

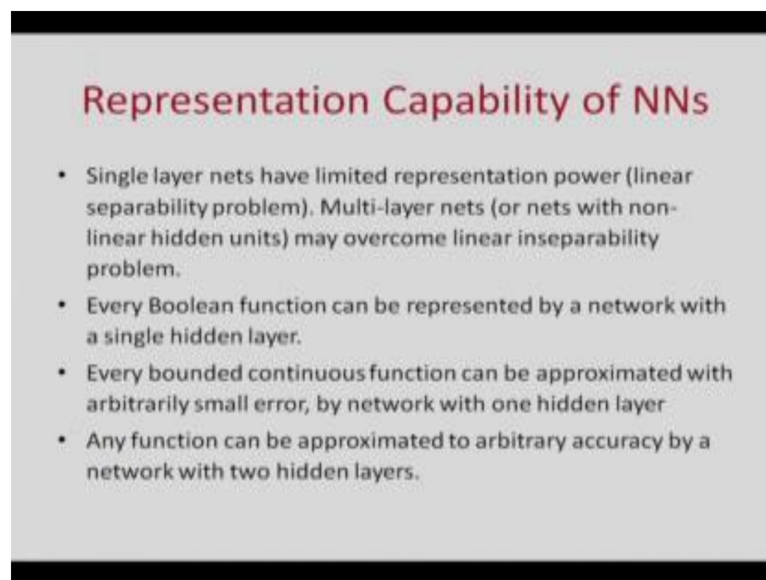
But, when we have the XOR function, which is 1, if exactly one of the inputs is 1 and 0; if both both of them as 0 or both of them and 1 that is XOR function and as we can see there is no linear decision boundary that separates the 0 points from the 1 point. So, in order to represent this function we can go for multi layer perceptrons.

(Refer Slide Time: 05:37)

This is an example of implementation of the Boolean XOR function. So, we have initially we have the first layer we have 2 perceptrons; the first perceptron h 1 and h 2 and then the second layer we have 1 perceptron and together these three units. In 2 layers, they can represent the Boolean XOR function for certain combination of weights, for example, we can have the first the left unit left h 1 at the first layers to represent the OR function by putting the weights as 1 1 and minus 0.5.

We can have the second node at the first layer represent the and function by putting the weights has 1 1 minus 1.5 and we can have the node at the second layer implement, the final XOR function by setting the weights has 1 minus 1 minus 0.5, this is one example implementation of the XOR function by using, 2 layer perceptron XOR cannot be implemented by a 1 layer perceptron, but it can be represented by a 2 layer perceptron function.

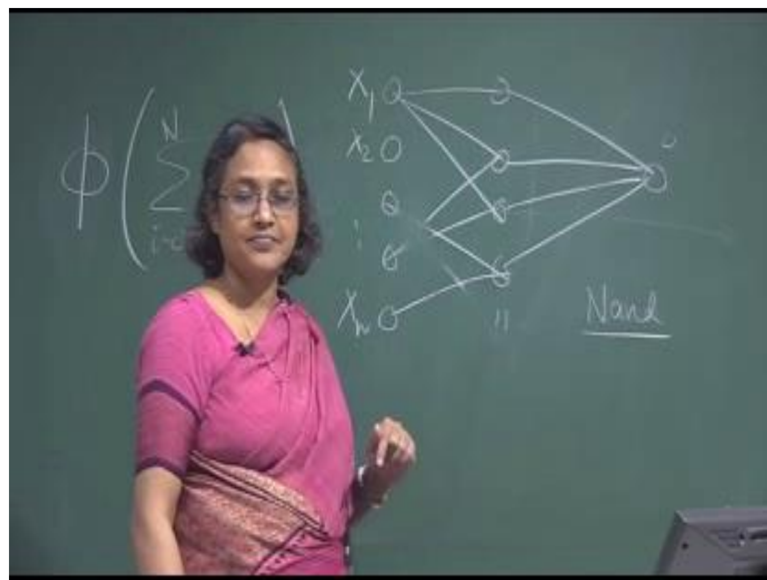
(Refer Slide Time: 07:00)



Now, in general if you look at multi layer neural networks, we can say this thing about the representation capability of neural networks if you have single layer neural networks they have limited representation power and we have already discussed they can represent linear decisions surfaces and therefore, if the examples of 2 classes are linearly separable then only they can represented by a single layer perceptron.

If you have non-linear functions you have to go for multiple layers and as you can see if we had only linear units combination of linear units could be at another linear unit. So, in order for multi layer neural networks to represent non-linear of function it is important that the functions implemented at the individual units are non-linear that is why we go for non-linear units either threshold unit or sigmoid unit or tanage unit or so on. Now, when we go for multi layer network, if we go for a 2 layer network, suppose this is the input unit and we have 1 hidden layer.

(Refer Slide Time: 08:29)



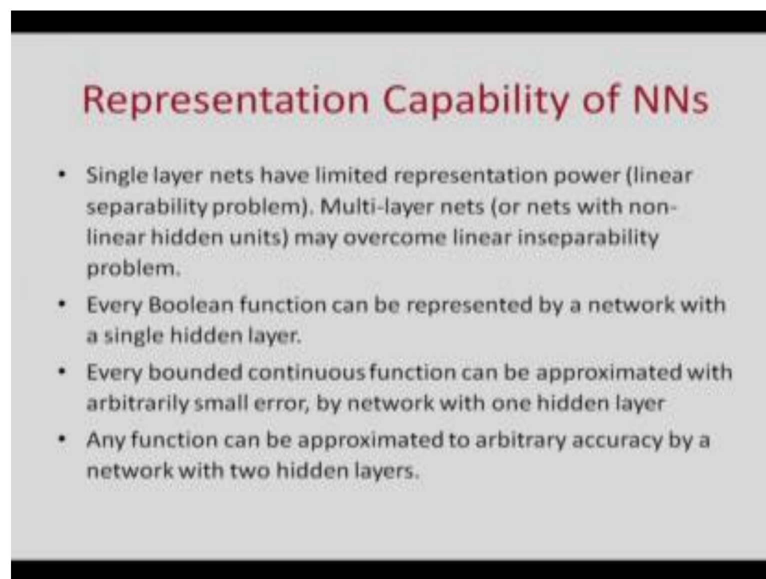
So, in a neural network, this is the input x 1 x 2 x n are the inputs and this is the output and we can have 1 or more hidden layers. Suppose this is the first hidden layer comprising three nodes right and we can have connection from the input to the first layer and we can have connection from the first layer to the second layer.

So, such a network this is the network with 1 hidden layer if we take a network with 1 hidden layer it is normally called a 2 layer neural network, such neural networks can represent all Boolean functions all Boolean functions can be represented by neural network with a single hidden layer. It is easy to see that it is possible because you may know that any Boolean function can be represented using Nand gates, using 2 layer of

Nand gates and I leave it for you to figure out the we can have a single layer perceptron represent the Nand function.

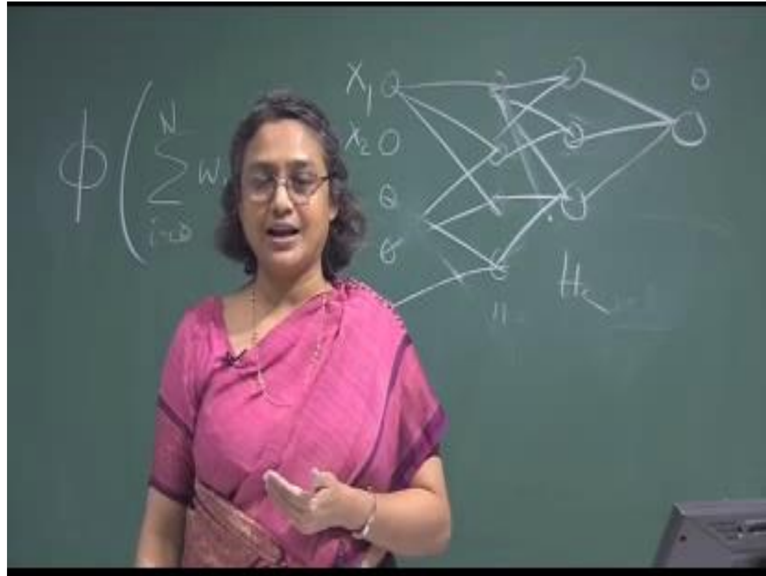
We have earlier seen that neural network can represent a single layer perceptron can represent the and function you can I leave it you as an exercise to see that it can represent the Nand function which is the inverse of the and function and by cascading 2 layers of Nand you can represent any Boolean function and any Boolean function can therefore, be represented by a neural network with a single hidden layer. So, this is quite obvious.

(Refer Slide Time: 10:27)



Secondly we can say every bounded continuous function can be approximated with arbitrarily small error by neural network with 1 hidden layer. So, not just Boolean function if you take a continuous function if that continuous function is bounded right that is it does not go to infinity, it is within a bound then any continuous function can be approximated by arbitrarily small error using single hidden layer neural network, but if you have a neural network with 2 hidden layers like this.

(Refer Slide Time: 11:12)

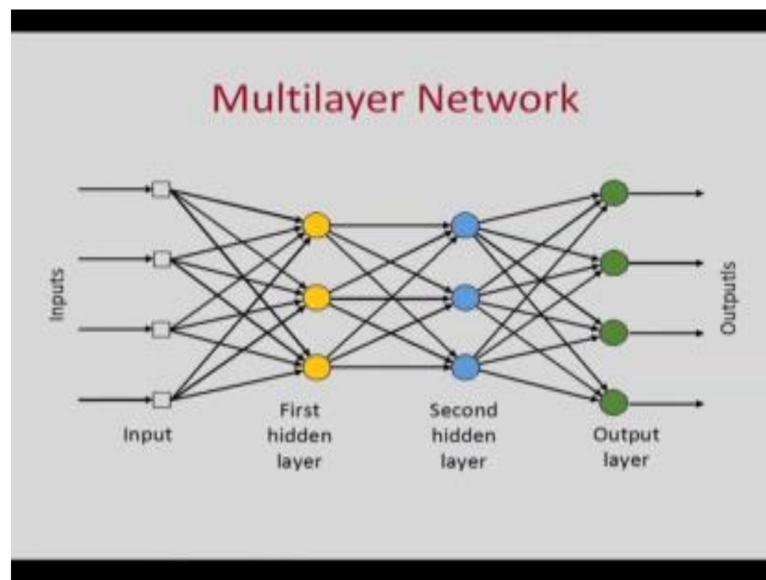


So, this is the first layer this is the second layer h 2 and there is a connection from the nodes in h 1 to the nodes in h 2 and then the there is the output. So, this is called a 2 hidden layer neural network it can be shown that any function at all can be approximated to arbitrary accuracy by a network with 2 hidden layers if you are using a network with 2 hidden layers such a network can represent any arbitrary function which is a very powerful statement; however, where is the cache just because given network can represent a function does not mean that the function will be learnable in the sense that as we will see in a neural network we do not know. So, we say that there exist neural networks which can represent this function, but that neural network comprises of a number of nodes in the different layers and that number of bits right.

So, we know that a function can be represented by a 2 hidden layer neural network, but we do not know how many nodes we should put what should the weights p and we do not know how may nodes we will put. So, that to figuring how to how many nodes we will put and what would be the weight may turn out to be hard for different problems and there when by said that any Boolean function can be represented by a network with 1 hidden layer I did not mention any thing about the number of nodes that you require they can be some Boolean function for which the number of nodes that you require can be very large. So, just because a function is representable may not mean that it is learnable.

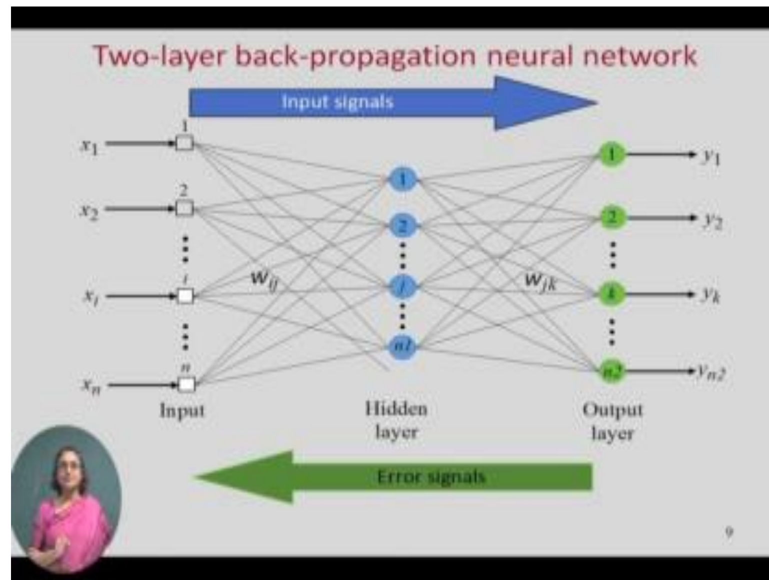
Now, we will see how we can learn in multi layer neural network using the back propagation algorithm now if you look at the slide this is the schematics of a multi layer neural network. So, we have the inputs we have the first hidden layer.

(Refer Slide Time: 13:25)



So, this shows at 2 hidden layer neural network the input the elder nodes are the first hidden layer the blue nodes are the second hidden layer and the green nodes are the output layer.

(Refer Slide Time: 13:38)



Now, as we have earlier discussed that when you give the input and you have observe the output and if you have a training set the training set will tell you for a given input what should be the ideal output and from the training set you can find out what is the error for a neural network unit for a particular input and we can update the weights. So, that this error is reduced. Now, the error is only observed at the output layer. So, if you have a neural network.

(Refer Slide Time: 14:13)

Where this is the input layer and this is the output layer and these are some hidden layers and there is connectivity between this and this, this and this, this and this for the nodes at the output layer we can find out the error for a given input. So, if I take the first input x 1 we can find out what is the output y 1 and we can find out what is the output that we are getting using the neural network. So, we can find out at every node for that given input what is the actual error and we can try to change the weights. So, that the error becomes small, but the error can only be observed that the output, we do not know for a training example what should be the value of a node here or a node here right.

So, what we are going to do is that the error that we find at this layer we are going to back propagate the error and estimate the error at the inside hidden layers we are going to take the error which we observed at the output layer back propagate the error to the previous layer. So, we say that the error here is because of the error which was computed here. So, the blame of the error here specially depends on at this node, which in turn depends on the error at these nodes. So, we will take this error and back propagate it to the other nodes from which it takes input and if you look at the weight of this edge and the weight of this edge if this has a higher magnitude of weight this node has higher contribution here if it as a smaller magnitude of weight it has a smaller contribution here.

So, when we portion error backwards we apportion the error proportional to the weight if the edge has larger weight we put larger error we apportion to that node. So, back propagation works in this way when we apply the neural network on a particular input the input signal propagates in this way, right input signal gets computed in this way. So, that we can get the output, but when we find the error we find the error at this layer and the error is back propagated to the previous layers and based on the notional error out after back propagation based on that notional error we do the weight updating at this layers.

So, here we update the weights based on the directly observed errors after we have back propagated the error we find the notional error at this level and based on that we change these weights again we back propagate this error further here and based on that we change these weights. So, that is why we call this method back propagation back propagation is a method to train multi layer neural network the updating of the weights of the neural network is done in such a way. So, that the error observed can be reduced the error is only directly observed at the output layer that error is back propagated to the previous layers and with that notional error which has been back propagated we do the weight update in the previous layers.

Now, in the last class we have looked at the derivation of the error derivation of the update rule of a neural unit based on are the error right. We saw how we could use gradient descent to find out the error gradient at a unit and we could change the error based on going to the negative of the error gradient just to recapitulate.

(Refer Slide Time: 18:41)

Derivation

- For one output neuron, the error function is

$$E = \frac{1}{2}(y - o)^2$$

- For each unit j, the output o_j is defined as

$$o_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

The input net_j to a neuron is the weighted sum of outputs o_k of previous n neurons.

- Finding the derivative of the error:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

If you have 1 output neuron the error function is given by E equal to half y minus o whole square for a particular input y is the expected out y is the goal standard output o is the actual output. So, y minus o gives you the error. So, half y minus o whole square is the particular measure of error that you are using.

Now, for each unit j the output o j is defined as o j equal to the function phi applied on net j when net j is the sum of the weighted sum of the units at the previous layer. So, net j is sigma w k j and phi is the non-linear function that we are using as we have mentioned we could use phi as a Sigmoid function or Tanh or Relu or some such function. And w k j corresponds to those edges which are coming from the previous unit to this unit. The input net j to a neural is the weighted sum of outputs o k of the previous n neurons which connect to this neuron and following the method which we followed in the last class now given this we can find out given that is formula E equal to half y minus o whole square and o is as it is defined here we can now try to find out the derivative of this error function with respect to the different weights.

So, as we have n different weight correspond n plus 1 different weights corresponding to the previous inputs and the bias we can take the partial derivative of this error function with respect to each weight. And this where quantity partial derivative of the error

function with respect to w i j which is 1 specific weight can be re written as by the chain rule del e by del o j times del o j by del net j times del net j by del w i j right. So, net j equal to sigma w k j o k and o j equal to phi of net j and we can write this del e by del w i j in this spot.

(Refer Slide Time: 21:27)

For one output neuron, the error function is $E = \frac{1}{2}(y-o)^2$

For each unit j, the output o_j is defined as

$$o_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$= \sum_l \left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_{z_l}} w_{jz_l}\right)\varphi(net_j)\left(1-\varphi(net_j)\right) o_i$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

with

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - y_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron} \\ \left(\sum_Z \delta_{z_l} w_{jl}\right) o_j (1 - o_j) & \text{if } j \text{ is an inner neuron} \end{cases}$$

To update the weight w_{ij} using gradient descent, one must choose a learning rate η.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Now,. So, this is what we have re-written here this is the error function this is o j and del e by del w i j this is what we did in the previous slide now this can be written as del e by del o j can be written as summation over l del e by del o l del o l by del net z l times w j z l which corresponds to this output is coming from the other a other outputs other nodes which are index by l and this is phi of net j into 1 minus phi of net j, which is the corresponding to the sigmoid function and this is o i this follows from the derivation that we worked out in the last class.

So, simplifying we can find out at del e by del w i j is delta j o i where this quantity is called delta j right the detail derivation we have done in the last class. So, del e by delta w i j is delta j o i where delta j comes from the nodes of the next layer from the errors in the layer is back propagated to the previous layer where delta j is given by del e by del o j del o j by del net j which is equal to o j minus y j into o j times 1 minus o j if j is an

output neuron if i am at the last layer otherwise it is recursively computed as sigma over z delta z l w j l times o j times 1 minus o j if j is a neuron in the hidden layer.

So, delta j can be computed directly at the output layer and once you have computed delta for each of the output units, you can compute of the previous unit after you have computed all the deltas of the previous unit you can compute the delta of two previous unit. So, that is called the back propagation and based on that this is the recursive computation of delta starting from the last or the output layer and going 1 level backward up to the beginning. Now, once you are figured this out you know what is del e by del w i j and then you change the weights using gradient descent delta w i j equal to minus eta del e by del w i j right. So, this is the gradient this eta is the learning factor small eta means slow convergence large eta means faster rate and minus because we are doing gradient descent.

(Refer Slide Time: 24:28)

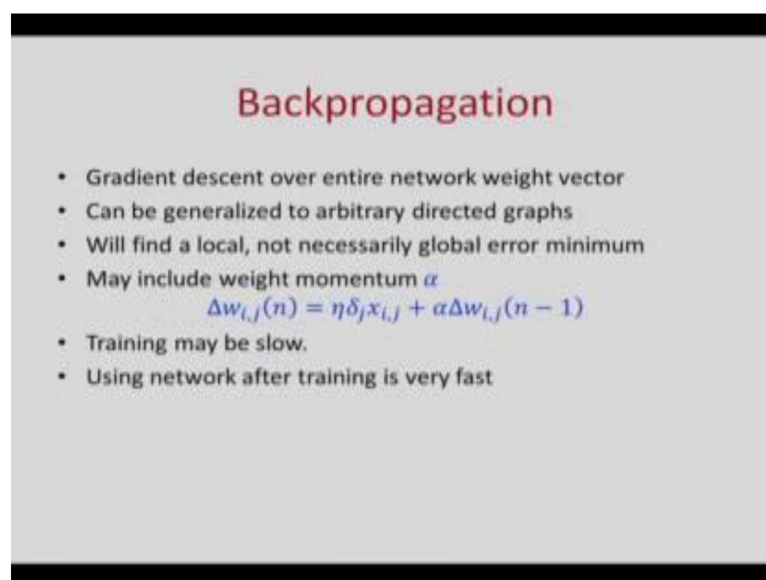
So, based on this we can write the back propagation algorithm which is actually very simple as is given in this slide, we take a neural network, we have a structure, we have 1 hidden layer or 2 hidden layer whatever you want you decide the number of layers in the neural network number of units in each layer and you do connection from this input to the first lead in layer first to the second, second to the output and then you have a number

of weights initially you initialize all the weights to small random numbers after that you carry out an iterative process which is given as here until satisfied what you do is you input you have a set of training examples you input the first training example to the network and compute the network output.

So, you give x you find 1 now for each you find o you get x 1 you find o 1 give x 2 you find o 2 now for each input unit k you may have only 1 output or multiple outputs. So, where each output unit k you compute delta k at the output layer as o k into 1 minus o k into y k minus o k then you go to the previous hidden layer for each hidden unit h you compute delta h as equal to o h into 1 minus o h into sigma over w h k delta k for all k which are in the outputs now after that you update each network weight w i j as w i j is w i j plus delta w i j and delta w i j is minus eta delta j x i j as we have already seen.

So, this is the back propagation algorithm we have an input we have the output that we get from the network and we have the target output we find the error from the target output based on that we update the weights we back propagate the weights. So, the previous layer by propagating the delta value and continue we continue for all the hidden layers. So, this is the back propagation algorithm, but it is very simple to implement.

(Refer Slide Time: 26:46)



So, in back propagation, we do gradient descent over the network weight vector and even though, the example, we have shown is for a layered graph we can do back propagation over any directed a cyclic graph the second thing to observe is that by doing back propagation we are not guarantee to find the global best we only get a local minima. So, we have a very complex error surface comprising of the weights at all the layers by doing back propagation we are updating the weights to do better and better and we continue doing it until the network converges, but when the network converges it will converge to a local minima which need not be a global minima and there are certain tricks that 1 can use to prevent getting trapped in a local minima.

For example, 1 such trick is to include momentum factor called alpha. So, the idea is that if you are going and trying to hit local minima. You try to prevent that by maintaining the previous direction of movement by the general direction of movement you do not want to deviate and get start. So, momentum what it does is that when you change delta w i j you not only look at eta delta x j x i j which we have derived earlier, but we also keep another factor which keeps track of the direction of weight change at the previous iteration delta w i j n is the weight change at the nth iteration which is equal to eta delta j x i j plus alpha times direction of weight change in the previous iteration.

If you apply the momentum training may be slow, but you are less likely to hit a local minima or bad local minima, but 1 thing to note is that in neural network when you use multiple layers even if training is slow after you have learn the weights applying the neural network is very fast.

(Refer Slide Time: 29:07)

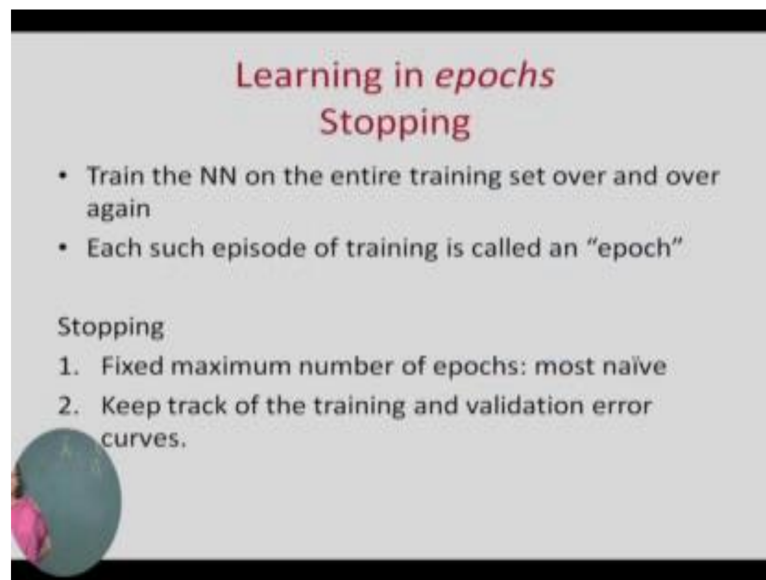
Now, there are few other observations, I will make when we do the weight update we can do a batch update that is given a particular configuration and given a set of training examples with respect to the all the training examples we can compute the partial derivative of the error and find the best way of updating it or we can do it for 1 input at a time.

So, the first method is called batch gradient descent. The second method is call the stochastic gradient descent there you take 1 input at a time based on that you change the weights now stochastic gradient descent is less likely to gets stuck in a local minima because if there is a local minima, it is unlikely that for all examples it will be that minima. So, first if you are stochastic gradient descent the neural network is more likely to get towards the global minima and there is something in between which is called mini batch gradient descent. So, instead of taking all the examples at a time or a single example at a time you take a batch of examples at a time and with respect to that you do gradient descent.

So, batch gradient descent calculates outputs for the entire data set accumulates the errors then back propagates and makes a single update. It is too slow to converge and it make gets stuck in local minima stochastic or online gradient descent on the other hand

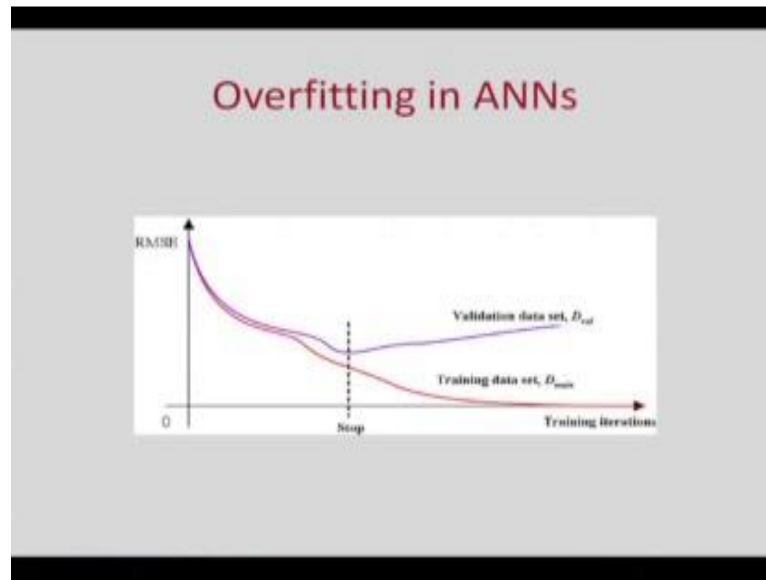
will take 1 training example at a time with respect to that it will find the error gradient it converges to solution faster and often helps get the system out of local minima and in between we have mini batch gradient decide.

(Refer Slide Time: 31:08)



Now, the entire training process can be divided in to epochs. If you have a number of training example 1, it epoch will look at all the training examples. Once then we will have the second epoch then the third epoch etcetera. When we are learning an epochs when do you stop? So, we keep training the neural network on the entire training set over and over again and each episode is called an epoch and we can stop where the training error is not is getting saturation or we can use cross validation while we are training the neural network, we can also keep validating it on a held outside and when we see that the training and validation errors are closed then we can stop or we can stop when we reach a maximum number of epochs.

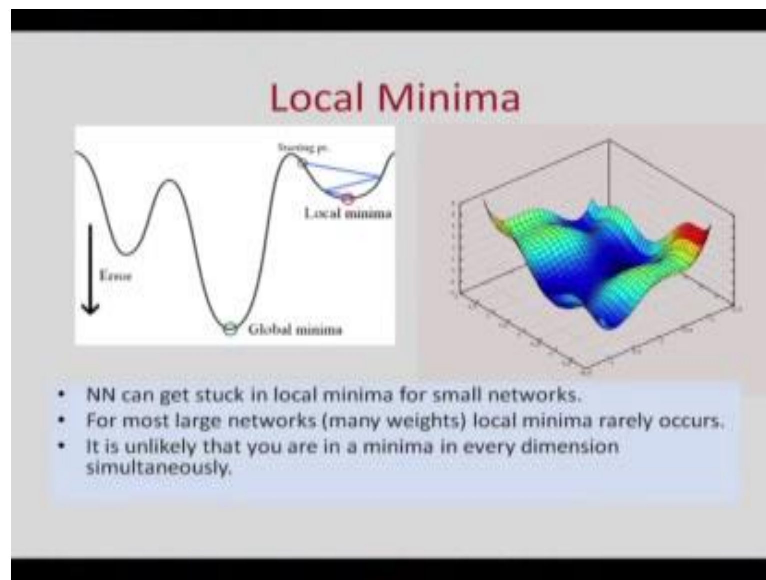
(Refer Slide Time: 32:04)



Now, in neural networks like other machine learning algorithms over fitting can occur and this over fitting is illustrated by this diagram. So, on the x-axis, we have the number of iterations and the y-axis, we have root mean square error as is typical of many machine learning algorithm as we increase the number of iteration the error on the training set keeps on reducing and it may even become 0 may or may not become 0, but the error on a held out set typically will initially decrease and then it will increase were over fitting has occurred right.

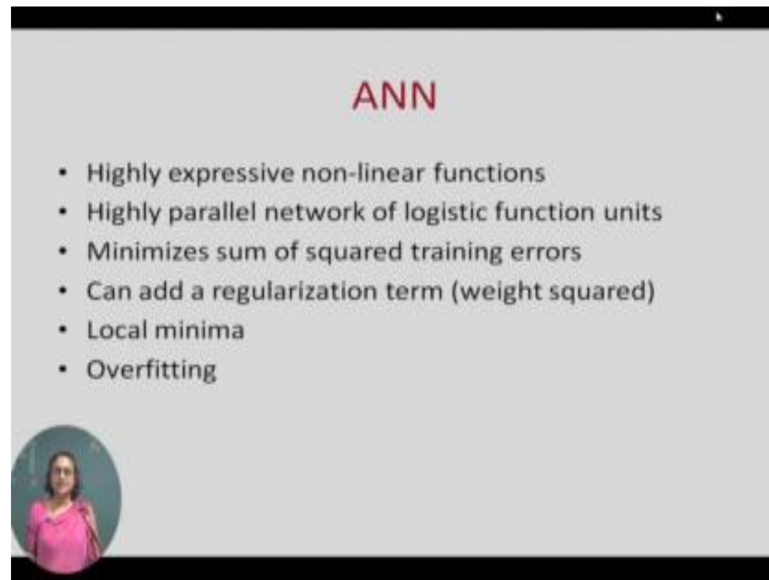
So, this is the zone where over fitting has occurred. Ideally you should stop before the validation error starts to increase. So, if you can keep track of the validation error you will know that this is the place where you must stop and not continue any more iteration because beyond that the network is likely to over fit and the accuracy of the network will go down.

(Refer Slide Time: 33:17)



So, this picture illustrates local minima as we said neural networks gets can gets stuck in local minima for small networks and we also said that if we use stochastic gradient descent it is less likely do get stuck in local minima, but in practice when you have a large network with many weights local minima is not. So, common because we have many weights it is unlikely that and you are doing every weight separately it is unlikely that the same local minima will be the minima of all the weights.

(Refer Slide Time: 33:51)



So, in conclusion we can say the artificial neural network, let us you use highly expressive non-linear function that can represent all most all functions. It comprises of a parallel network of logistic function units or other types of units are also possible the principle works by minimizing the sum of squared training errors there are also neural networks with different other laws functions, but we will not talk about it in this class here we have looked at neural networks to minimize the root mean square error we can add a regularization term to a neural network which I did not talk about.

So, what you can do is that you can write to prevent the weights from getting large by penalizing networks where the weights have large values by adding regularization term neural networks can get stuck in a local minima and it may exhibit over fitting.

With this, we come to a conclusion in the next class. We will give a very brief introduction on deep learning.

Thank you very much.