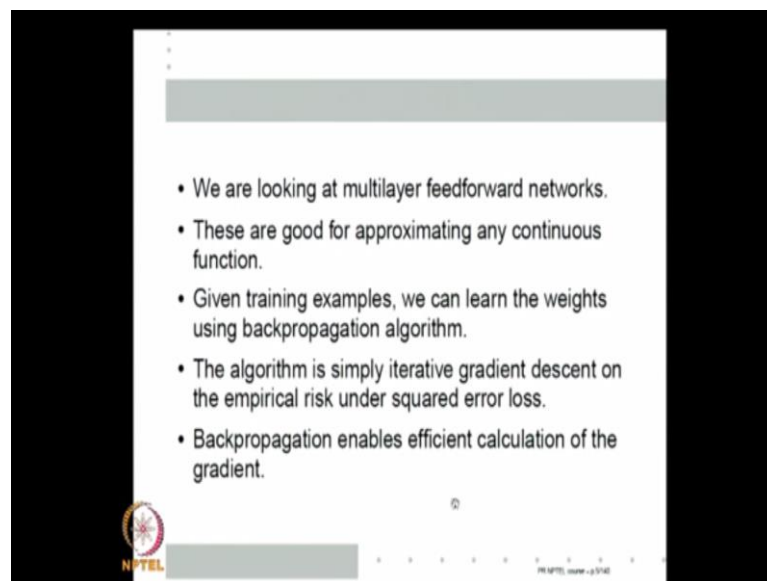


Pattern Recognition
Prof. P. S. Sastry
Department of Electronics and Communication Engineering
Indian Institute of Science, Bangalore

Lecture - 29
Feedforward networks for Classification and
Regression; Back propagation in Practice

Hello and welcome to this next lecture in pattern recognition course. This lecture will complete our discussion on multi layer feed forward networks. So, just to recall the context; we have been discussing this so called multilayer feedforward neural networks these are networks where each unit takes a weighted sum of its inputs such that it passes it through an activation function a sigma either a tan hyperbolic activation function and that is how output is calculated. We have multiple layers in networks and layer by layer we will calculate the output that is how inputs get transformed into output.

(Refer Slide Time: 00:59)



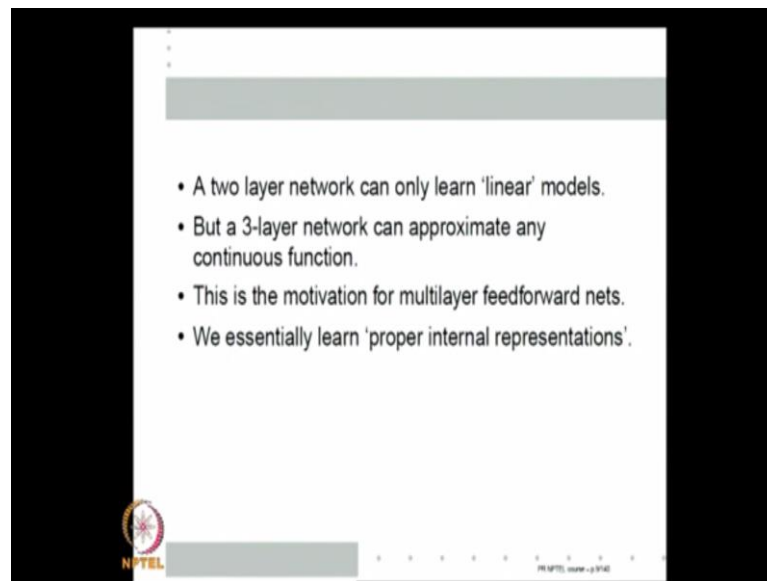
So, as we saw these multilayer networks or good as model for approximating any continuous function right last time we actually saw a theorem which told you that if I want to approximate any continuous function it could be as non-linear as we want but, any continuous function on a compact sets in \mathbb{R}^m . Thus continuous real valued function on compact sets in \mathbb{R}^m can be approximated by networks of this kind. So, the idea is that

given training examples, we can learn the weights using back propagation algorithm. So, because we know that the structure is good enough for representing approximating most functions we choose some structured network where we could have still have to decide how to choose but, we choose let us say one hidden layer in sufficient nodes hidden nodes. Then, we can give the training examples we use the training examples to learn the weights on for that we seen the back propagation algorithm.

The back propagation algorithm is nothing but, it is an iterative gradient descent the function we optimizing actually minimizing is this some of squared error that is the empirical risk under squared error loss function which is simply sum of squares of the errors what that I mean for each input pattern I calculate the network output, the desired output, take the difference square it at the squared the error. We submit over all the training patterns this will be some function the weights the network and we are minimizing this were doing gradient descent on function to find the optimal weight. So, the trying to find weights to minimize the squared error the interesting thing is that the back propagation algorithm as you saw last time, it enables for a very efficient calculation of the gradients.

To actually do the gradient descent on this function, we need to calculate all the gradients that the gradient of the function with respect to all the weights and the back propagation algorithm by a computation that is similar to a forward computation of calculating the outputs of the network given the inputs by computation, which has about as much computation complexity what we called the back propagation it allows should calculate all the partial derivatives. Partial derivatives for the either with respect to all the weights so, that is how back propagation allows us a efficient way to implement the gradient descent.

(Refer Slide Time: 03:17)



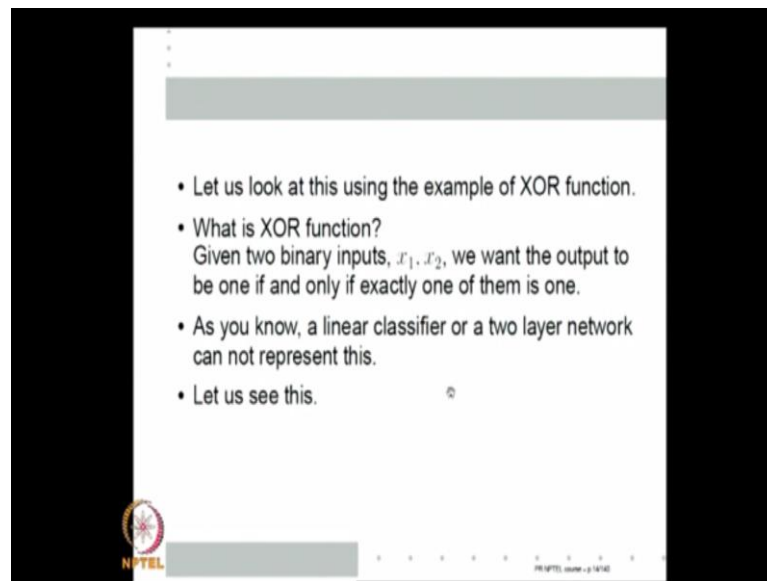
Now, we also seen that in our undone notation the first layer is the input layer, last layer is the output layer. So, if I have only 2 layers then the no hidden nodes. So, is like a perceptron or an adrenaline. So, these are the linear model set of consider earlier. So, a 2 layer network can learn only a linear model where as the moment I put a 3 layer network as we saw in theorem in the last class. I can approximate any continuous function essentially by a putting one more layer am I allowing myself to, I am allowing myself to a family of functions which can approximate any continuous function.

Now, as we said this is the motivation for considering multilayer feedforward network of course, even though theoretically one hidden layer is enough in practice. We may need more than one hidden layer but, in any case while a 2 layer network in our notation what will be a 2 layer network is can learn only nearer models in moment I have I have a hidden layer I can approximate any continuous function if there are sufficiently many hidden nodes.

So, in that sense this forms a very nice parametrized class of non-linear functions. Essentially as we saw by putting a hidden layer, the input is first getting transformed into the hidden layer on to the output hidden layer and then these outputs of what go to the output node. So, if we look at the output layer and the layer just before the output layer those 2 together will be a 2 layer network. So, that is essentially a linear model but, that linear model the impush to that linear model or not the original inputs but, the outputs of

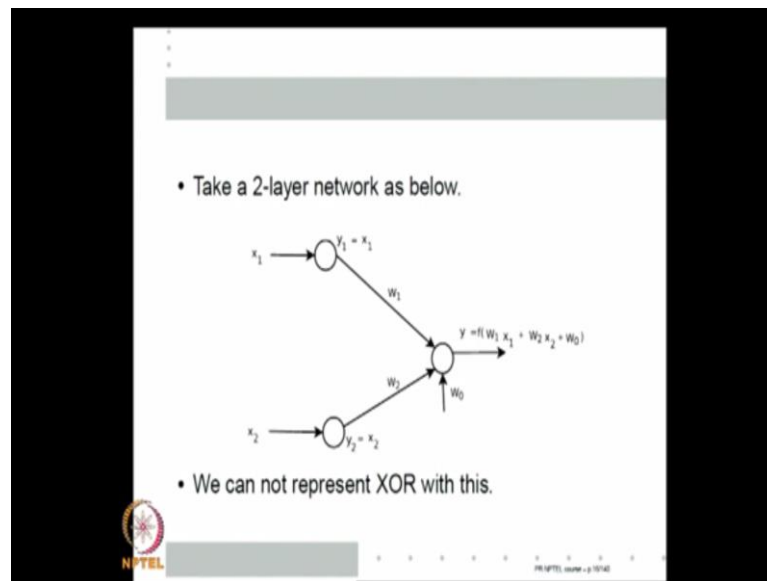
that layer just before the output layer. So, what the network is doing is essentially learning a proper internal representations that is while access the actual given input I transform into some of that representation as the output of the layer just before the output layer in that representation a linear model is good enough. So, that is essentially what we are doing. So, let us take a little more let say one example to understand what we mean by proper internal representation.

(Refer Slide Time: 05:32)



So, we will we will look at one function to understand this let us say we take the XOR function when we did the linear models for classification regression we said say XOR is one example of a classification problem which is not linearly separable. And hence a linear model not good enough what is the XOR function it has 2 inputs and 1 output. So, given 2 binary inputs x_1, x_2 we want the output to be one if and only if exactly one of them is one. If both of them are 0 or both of them are 1 I want the output to be 0 but, if only exactly one of them is 1 then I want the output to be 1. As you already know a linear classifier or a 2 layer network cannot really represent this right this is not linearly separable. So, linear model kind exactly represent this function.

(Refer Slide Time: 06:31)



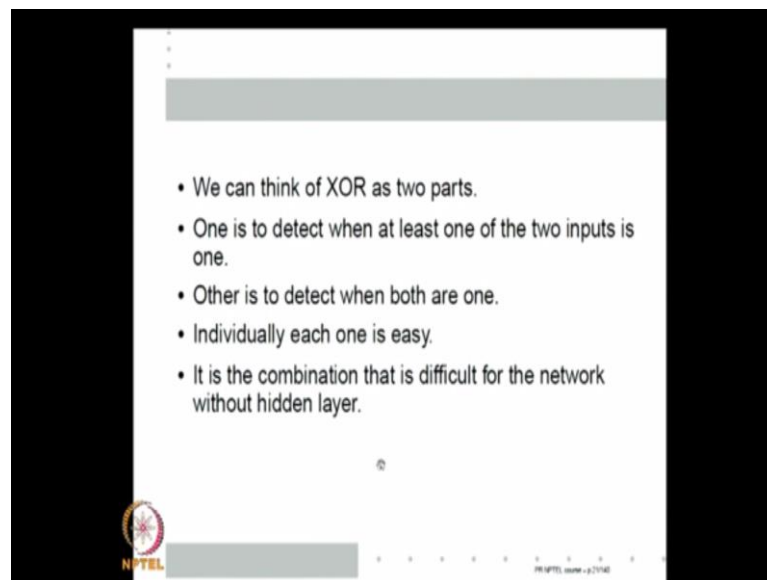
Let us try and understand in some sense a it is a kind of proof not really proof a possibility argument has to why a truly a network cannot represent this. Let us take this a 2-layer network as $x_1 \times x_2$ these are the input nodes so the output of this node is simply x_1 the output of the node simply x_2 the 2 weights on then there output node output node can have a bias so the final output I can get is f of $W_1 \times x_1$ plus $W_2 \times x_2$ plus W_{naught} .

Now, when the 2 inputs are 0 the outputs are 0 right at that time let us say y should be low means because I am using sigmoid function if the net input is small then output will be small. So, let say we put our weights W_1 W_{naught} such that when this are 0 of course, W and W_{naught} are does not matter because that will be 0 so it only be W_{naught} so W_{naught} is sufficiently negative. So, that when both the inputs are 0 my y will be close to 0.

Now, suppose x_1 becomes one x_1 becomes one i want the output to go to one now what I changed in that input because x_2 still 0 this is still 0 the old W_{naught} is still there. So, the only extra one is because this output in one now i am getting an extra W_1 in the net input. So, f of W_{naught} should be close to 0 as double numbers should be some negative number but, f of W_1 plus W_{naught} should be close to one so W_1 should be sufficiently positive to overcome this minus W_{naught} right. In the same way suppose x_1 is 0 but, x_2 is one once again y should go up so W_2 should be sufficiently positive.

So, that if the net input is simply W_2 plus W_0 of course, W_2 W_0 should also be sufficiently high. So, W_2 should also be high and positive and high enough to overcome the negative thing of W_0 I need W_0 to be negative because even both of them 0 I need y to be 0 so given this if both x_1 x_2 are one now I get both W_1 and W_2 W_1 till W_0 itself is enough drive my sigma it is through out one if I add some more positive quantity it come closer to one there is no way it can when both x_1 and x_2 are one the output can becomes suddenly 0 because from when 0 when both of them when both of them are 0 the output is 0 any one of them is one the output has to go to one. So, both W_1 and W_2 weights have to be sufficiently positive this implies that when both of them are one I have no choice but, driving the output to one this is the reason why such a network cannot represent XOR ok.

(Refer Slide Time: 09:21)

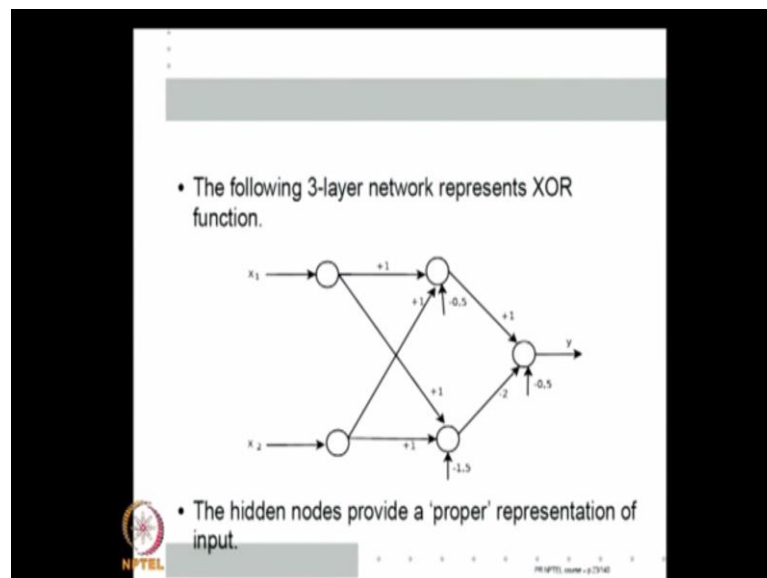


On looking at this the following we can think of XOR the function XOR having 2 parts one is to detect when at least one of the 2 inputs is one. The other part is to detect when both are one. Let say only want to detect when at least one of them in the 2 inputs is one that is a OR function out of the 4 possible inputs only in one case it should be 0 other 3 should be in one and hence that is linearly separable it is very easy to visualize you know a square where one corner is in one class the other 3 corners other class so is linearly separable similarly, detecting both are one is an AND function once again if I take the 4 corners of a square one corner is one class and 3 corners in the other class this is also linearly separable. So, detecting on at least one of the 2 inputs is one is easy in sense a

linear model will do detecting when both inputs are one that is also easy once again linear model will do individually each one is easy where is the problem the combination that is difficult for the network without hidden layer.

Basically, I have to detect the combination that at least one of the inputs is one but, not both are one right that combination is what is not freezable to be done with linear model. This combination can be done with a hidden layer because this is the adrenal representation essentially instead of representing $x_1 \times x_2$ as the 2 binary inputs if I can transform that inputs into one that flags whether are not at least one of them is one other that flags whether are not both are one but, this I can do in one in from inputs to hidden layer because that is like a small 2 layer network itself and it can learn any linear function. So, I can have hidden layers that can represent either this part or this part with that internal representation is very easy now for me to represent XOR because when this is one and this is 0. I want to flag one other wise no whenever this is one I want to flag 0.

(Refer Slide Time: 11:50)



Now, let us look at this here is one possible way these are the inputs nodes. So, they give you one and 0s here. So, look at this hidden node because both this weights a plus 1 you will get x_1 plus x_2 if both of them 0 let us say in this network because we are using sigma x for both the hidden nodes and the output node. So, let us say if the net input is minus 0.5 will assume that sigma it is close to 0 and if net input is 0.5 or more then the output is close to 1 that is all we want to get. So, both x_1 and x_2 are 0 then is only

minus 0.5 the net input so the output of this is close to 0. If at least one of them is 1 then it will be 1 net input will be 1 minus 0.5.

So, net input will be 0.5. So, output will be sigma net 0.5 that will be close to one if both of them are one it will be even closer to one. So, this hidden node is discovering whether at least one of the 2 inputs is one and this side because I increase the bias if both of them are 0 it will be sigma net input is minus 1.5 so input is very close to 0 if only one of them is one then net input still minus 0.5. So, it is still close to 0 only both of them are one then net input will be plus 0.5. So, it will be close to one.

So, this nodes output will be one only when both are one. So, this is essentially indicating whether both are one this is indicating whether at least one of them is one. If the outputs of both this nodes are 0 that means both are not one and not even one of them is one so is all 0s. So, when everything is 0 I want y to be 0 that is easy to achieve by having a negative bias minus 0.5 when output of this hidden node is one where output of this hidden node is 0 that means one of them is one but, not both then I want y to go one so I can put a positive weight on this if I put plus 1 here plus 1 minus 0.5 I will get a net input of 0.5 or more so the output will go will be high and other then this become one when this becomes one this will also become one obviously when this becomes one I am get giving minus 2 into net input here that will more than form this plus 1 right and hence the output will now, go to 0.

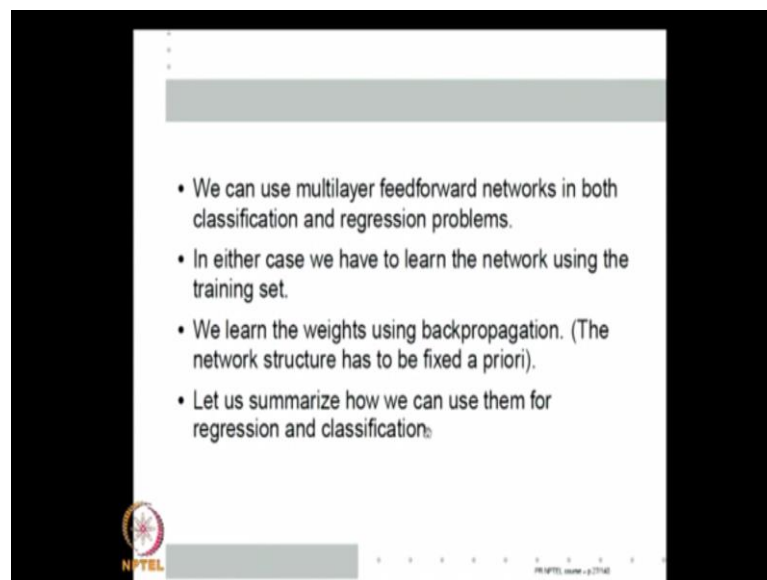
So, this is how the hidden layer cannot represent XOR it has written learned it has right internal represent of course, here we are not showing any learning I mean whether these this that be many other ways of representing this but, this is one kind of a internal representation that I can learn in this problem and the point of the example is that by having the right internal representation here. This is just a linear model and that linear model is enough because as the right internal representation for the now each of the internal representation (()) because of one layer can be land fit a bilinear model again that is what I am doing.

So, in all this network essentially they function because I can learn the right internal representation of course, if we actually take the squared error and do gradient descent whether you converse to these particular way it is a different issue that depends on the

how the gradient behaves how the gradient descent behaves of course, there other issues also for example, in suppose I have one 2 3 4 5 six seven eight nine weights. So, it is some nine dimensional weights pairs which I will be doing the optimization. Now, if I just switch this 2 nodes I call this node as the second node in this layer and this node the first layer if I interchange this node in this node what it means is what was W_1 and W_2 earlier will now become W_3 and W_4 right.

So, just by doing some permutation of the nodes I will be representing the same function but, in the gates space it will be different vector. So, obviously there will be many weight combinations which will be good right. So, actually that this is this kind of optimization is hard optimization because there multiple good medium are there but, we will come to whether the algorithm launch well or not but, always showing here that this is what is meant by internal representation and this is what we seek to learn we may learn may not learn depending on whether the gradient descent gives us good local medium or bad local medium but, the idea is that this is how 3 layer network multilayer network represents functions because they can represent the right internal representations ok. So, the hidden nodes here provide a proper representation of the input and hence the network as a whole can represent the XOR function ok.

(Refer Slide Time: 17:04)

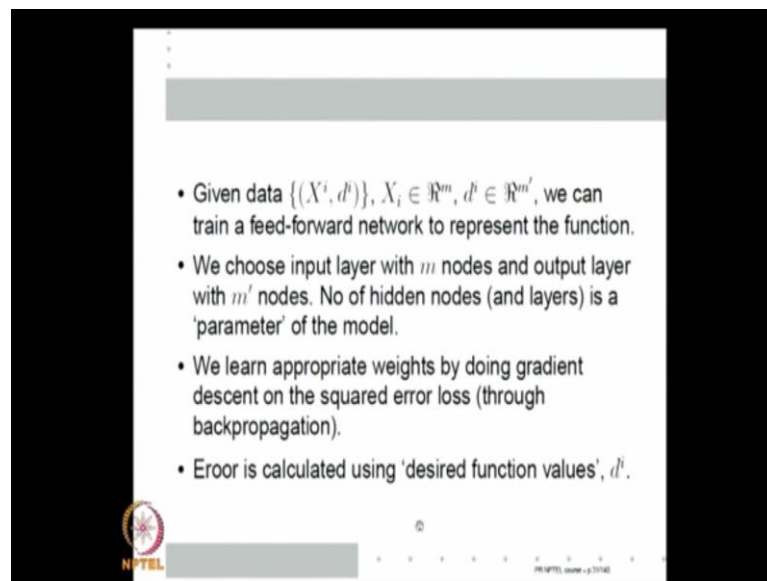


As move on so we can use multilayer feedforward networks both in classification regression problems and in either case we have to learn the network using the training set

right by we been what we have been doing also the course so given some training examples either for a classification problem or a regression problem I have to learn the model. So here we have to learn the network and learning the network is essentially learning the weights the network structure has to be fixed beforehand then I will learn the weights to minimize the squared error and we can learn the weights to using the back propagation.

So, let us quickly just see how I use it for regression how I use it for classification because we said that the network can arbitrarily approximate any continuous function. Of course, a classification function is not really continuous function so we have to ask how do I used it for classification and so let us start with how I use it for regression then we can see how to use it for classification also.

(Refer Slide Time: 18:10)

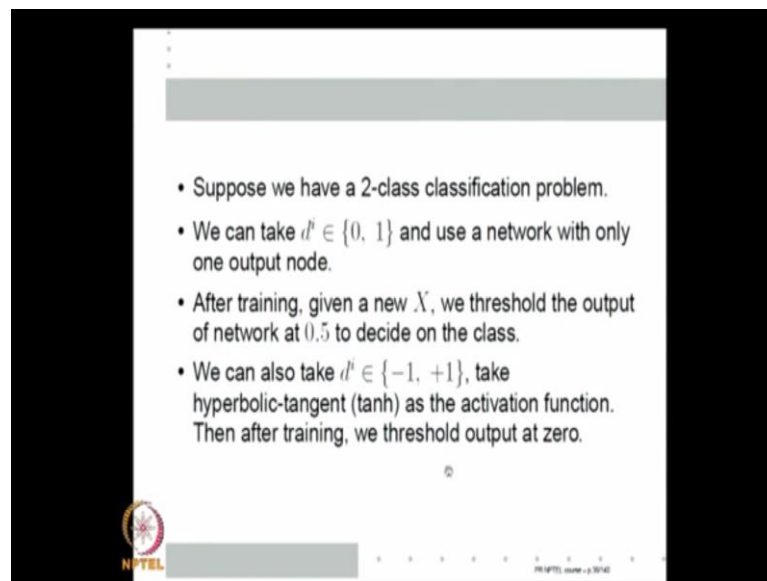


So, for a regression problem what I may given I am given data some X_i and X_i belongs to \mathbb{R}^m and d^i belongs to $\mathbb{R}^{m'}$. We need to train a feedforward network to represent the function this is what a regression problem is right. In general of course, m' can be one or more is one that is the standard regression consist otherwise you are we are learning a vector value function. What we do we choose an input layer with m nodes because inputs comes from \mathbb{R}^m we choose an output layer with m' nodes because these there outputs are m' dimensional and we have to put some hidden layers and hidden nodes we know theoretically one hidden layer is good enough but, practically you

may need more than one might be very often we may have need 2 and in each layer we need to decide how many hidden nodes we want these are arbitrary decisions and anyway that the parameters of the algorithm.

Once we have fixed a structure we can learn the weights but, doing gradient descent on the square error loss using back propagation this is how we learn the essentially to do the gradient descent we need to calculate the error on each training sample and in this particular case is absolutely no no difficulty in deciding what the error is because the desired inputs are given desired outputs are given if I give X_i as input you my training sample does me exactly what should be the output right. So, the error is always calculated using the desired function values namely d_i so this is how we can certainly learn functions.

(Refer Slide Time: 19:40)



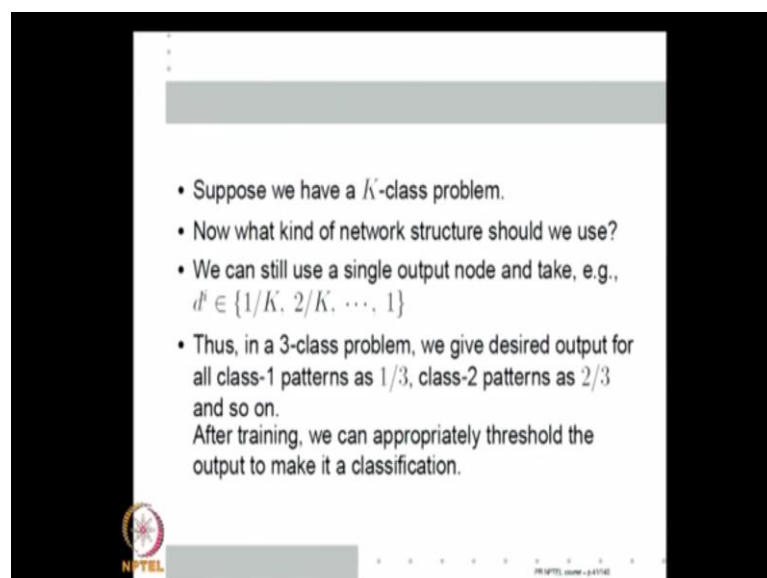
Now, let us consider classification case in the classification case I am just given feature vectors and class labels suppose I have a 2 class problem my class labels can plus one minus one can be one 0 whatever now what should I choose for 2-class problem that does not seem to much this thing we can think of a desired output set to be 0 and 1. So, I want to train my networks so that when I put a class one pattern the output is close to 0 and I put a class 2 pattern the output is close to one.

So, that why in the training sample whenever you give me class one pattern I said the

desired output is 0 whenever you give a class 2 pattern I said the desired output as one. Now, we can train the network of course, the network cannot actually represent the discontinuous function will classifier is a discontinuous function but, it will represent some continuous approximation to it but, after hence after training we expected learns a function which whenever you put a class one pattern the output will be close to 0 it may not be 0 but, will be closer to 0 than one and similarly, when you whenever you put a class 2 pattern your output will be closer to one rather than 0. So, after training and when you give me new X I can threshold the output at 0.5 to decide on the class the output is closer to one and I will say class one when output is closer to 0 I will say class 2 ok. Of course, this 0 on one is arbitrary I can as well use plus 1 minus 1 for example, why its plus 1 minus 1 my output should be go to both signs.

So, I will use hyperbolic tangent rather than sigmoid at the activation function where the act sigma hyperbolic tangent activation function I can use the desired output such plus 1 and minus 1 right. So, essentially to use the network as a classifier in a 2-class of case my input layer will have as many nodes as the feature vector dimension I am using only one output node. So, my desired output is a scalar and I am saying either use 0 and 1 at the desired output or minus 1 and plus 1 as desired output (()) 0 on one i use sigma little activation if I use minus 1 plus 1 I will use hyperbolic tangent as activation and if I use this obviously after training we threshold it at 0 instead of it 0.5 in this case ok. So, it is very clear how to use it for a 2-class classification problem.

(Refer Slide Time: 22:00)



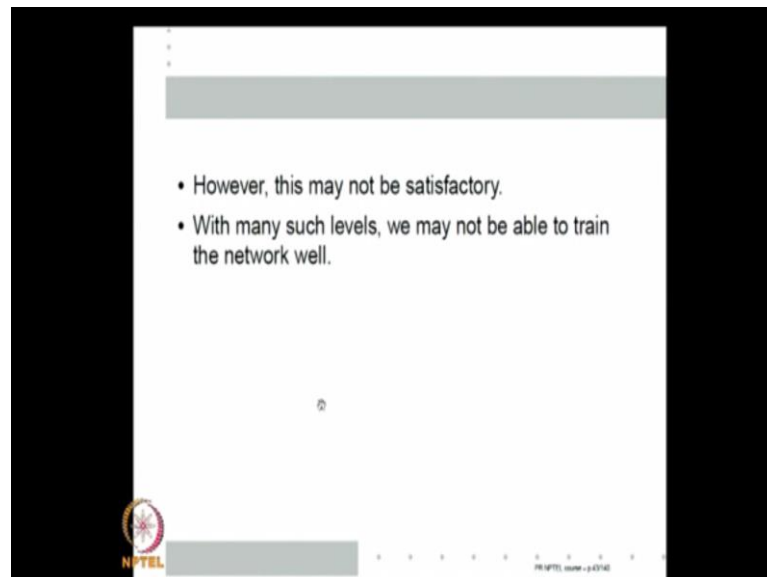
- Suppose we have a K -class problem.
- Now what kind of network structure should we use?
- We can still use a single output node and take, e.g., $d^k \in \{1/K, 2/K, \dots, 1\}$
- Thus, in a 3-class problem, we give desired output for all class-1 patterns as $1/3$, class-2 patterns as $2/3$ and so on.
After training, we can appropriately threshold the output to make it a classification.

NPTEL course - 47742

Suppose, we have a K - class classification problem then what do I do what should be the desired outputs now they can be many possibilities for example, we can still use a network whose input layer at the same dimension of the feature vector output layer is still uses only one node and I can give my desired inputs as one- k 2- k k by k . So, if I am basically the idea is that whenever I put class i pattern I want the output should be close to i by k right. So, for example, if I have a 3-class problem, for all class-1 patterns I will give the desired input as output as one-thirds for all class-2 patterns I will give as 2-thirds for all class-3 patterns as 3-thirds.

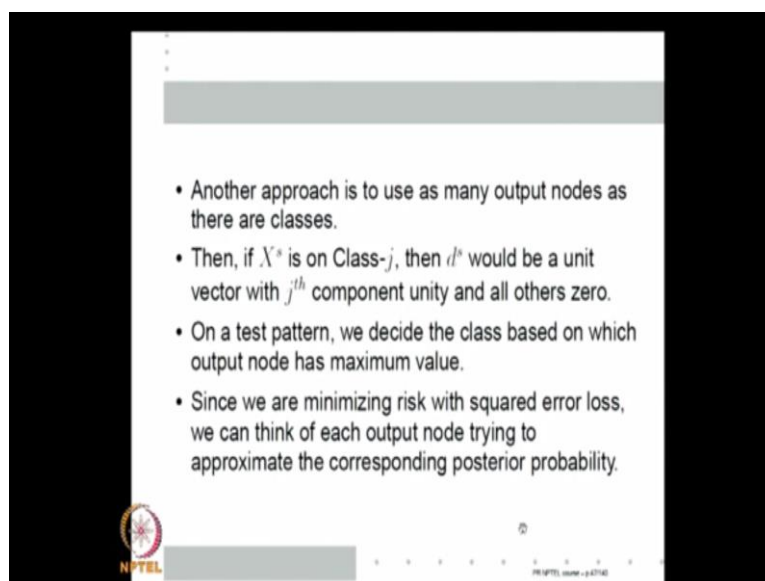
The idea is now the network is learn a function such that when I put a new vector x if it is of class-1 the output should be close to one-thirds the class-2 should be close to 2-thirds and so on. So, I take the actual output I ask is it closer to one-thirds as close to 2-thirds as closer to one and based on that I can make my decision right. So, after training you can approximate appropriately threshold the output to make a classification decision but, some of this does not look very nice right this one-thirds 2-thirds is very arbitrary.

(Refer Slide Time: 23:22)



And if I have too many levels it is not nice to expect the network to be able to do such fine estimation right as it is what you are teaching a discontinues function that function represent only a continuous function approximation and then you have this fine levels and this 1 by 3, 2 by 3 kind of business looks very arbitrary. What else can I do?

(Refer Slide Time: 23:47)



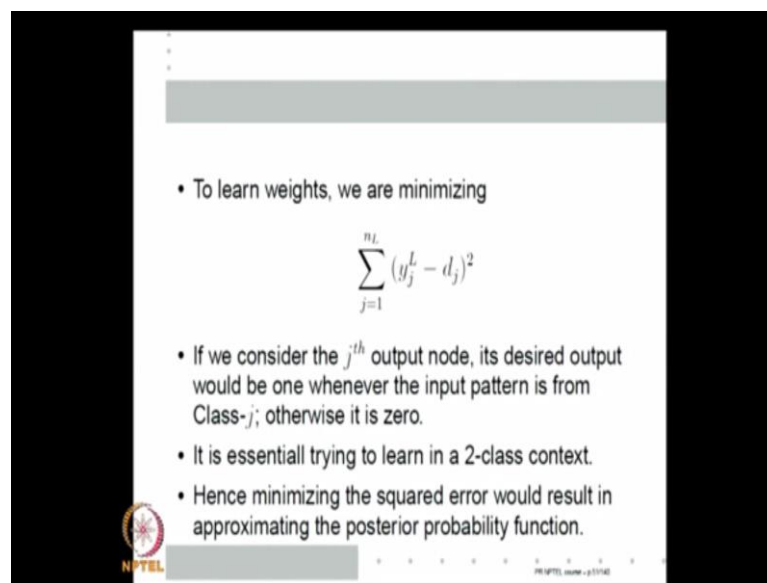
Here is another approach. Now, my input layer will have as many nodes as the feature vector dimension and the output layer will have as many nodes as there are classes if I am considering a k class problem I look k output nodes we have k output nodes that means my desired output when I give my training examples my desired output should be k dimensional vectors right. So, how do I give my desired output if a particular in example patterns X^s is in Class- j that the correspondence desired output d^s which should be a k vector will be a unit vector with j th component unity and all others 0.

So, if X^s is in Class- j d^s will be 0 0 0 1 in the j th position and 0 0 0 right d^s would be a unit vector with j th component unity and all others 0. So, this is how I specify the desired output for the j th (()). Now, this is interesting now essentially if you consider any one output node and say take the rest of the network connecting upto that output node that output node is essentially has a desired output of either one or 0 depending on whether the pattern present it is in class in that particular class or not right. So, this is like a proper generalization of the 2 class idea 2 k -classes. So, the test pattern we can decide the class based on which of the output node has the maximum value because we want the if it is in Class- j d^s is such that the j th component of d^s is one so whenever in Class- j j th output node should be closer to one.

Hence, when you give me a new test pattern after learning I calculate all the k outputs whichever output is closer to one means whichever output is a largest from some using

sigmoid activation function whichever prefer to one is saying say whichever output is highest value we will put the pattern in that class. As you shall see in the next couple of slides since actually we are minimizing squared error loss, as we have seen earlier with when we did linear regression we can think of each output node in this structure approximate in the corresponding posterior probability. As we seen in linear least square also, we are essentially train to appropriate a posterior probability. So, here when we use this kind representation the network is a allowing us to get good approximation of posterior probability.

(Refer Slide Time: 26:21)



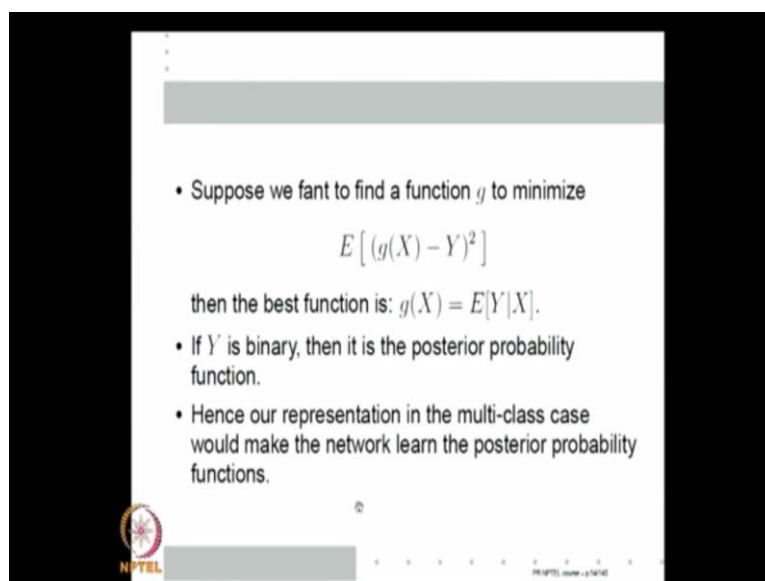
- To learn weights, we are minimizing

$$\sum_{j=1}^{n_L} (y_j^L - d_j)^2$$

- If we consider the j^{th} output node, its desired output would be one whenever the input pattern is from Class- j ; otherwise it is zero.
- It is essentially trying to learn in a 2-class context.
- Hence minimizing the squared error would result in approximating the posterior probability function.

See why? Essentially whatever we doing to learn the weights we are minimizing this. This is the output of the network the j th component of the output of the network this is the j th component that desired output d_j we take square sum over j this is what a minimizing. So, in a for a particular vector let say in some class all other components except this component will be one 0 ok see if we consider j th output node is desired output would be one whenever the input pattern is from Class- j otherwise it is 0. So, this is the reason so essentially it is still trying to learn in what can be called a 2-class context and this is the reason why minimizing the squared error would result in approximating the posterior probability function.

(Refer Slide Time: 27:09)



• Suppose we want to find a function g to minimize

$$E[(g(X) - Y)^2]$$

then the best function is: $g(X) = E[Y|X]$.

- If Y is binary, then it is the posterior probability function.
- Hence our representation in the multi-class case would make the network learn the posterior probability functions.

NPTEL

PR NPTEL course - p 14740

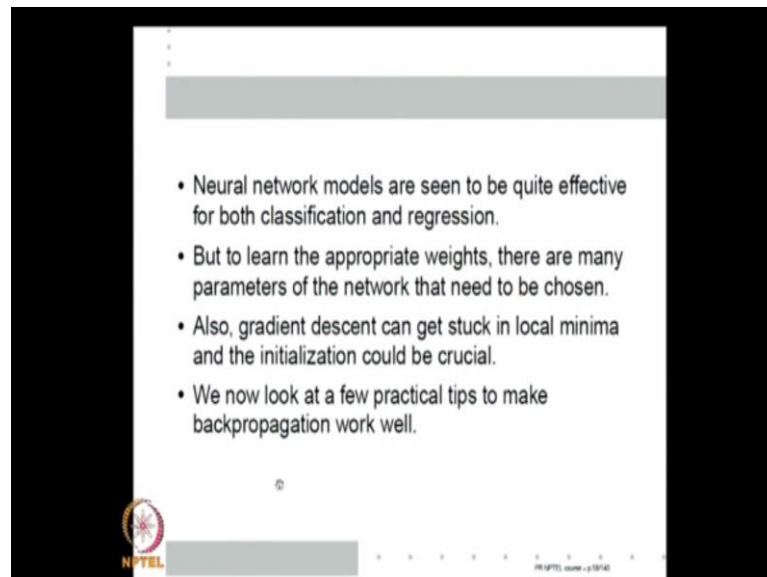
More precisely let say given to random variables X and Y I am trying to find a function g to minimize expected value of $g(X) - Y$ whole square by square error loss is simply an empirical risk of this risk this is the true risk of squared error so I can think as g as the network X as the input so $g(X)$ is the output of the network when X is given at the input so g is the network function Y is the desired output ok. Now, here if we are considering only one node so this square is what I am minimizing then we know that the best function g is expected value Y given X right this we have proved some time back.

See Y is binary the conditional expectation Y given, X is same as probability, Y is equal to one given X which is nothing but, the posterior probability function so the best function here is the posterior probability function. So, whenever I am minimizing this expectation the best g is the posterior probability function because this is what I am minimizing the network with the training inputs 1 or 0 for each one of the output nodes each output node is correspondingly approximating the each output node approximating the corresponding posterior probability right.

Hence our representation in the multi-class case that is when I put as many output nodes at their classes and then represent each class by k dimensional vector where j th component is one and all others 0 that representation would make the network learn the posterior probability function that is the reason so the first output node will learn the posterior probability function where the class one second output node will learn the

posterior probability function for class 2 and so on right and hence that the reason why if I take the max its essentially trying to approximate the base classifiers so in that sense this particular structure of having as many output nodes as there classes is quite good for multi class case when we want to use neural network of this kind ok.

(Refer Slide Time: 29:16)

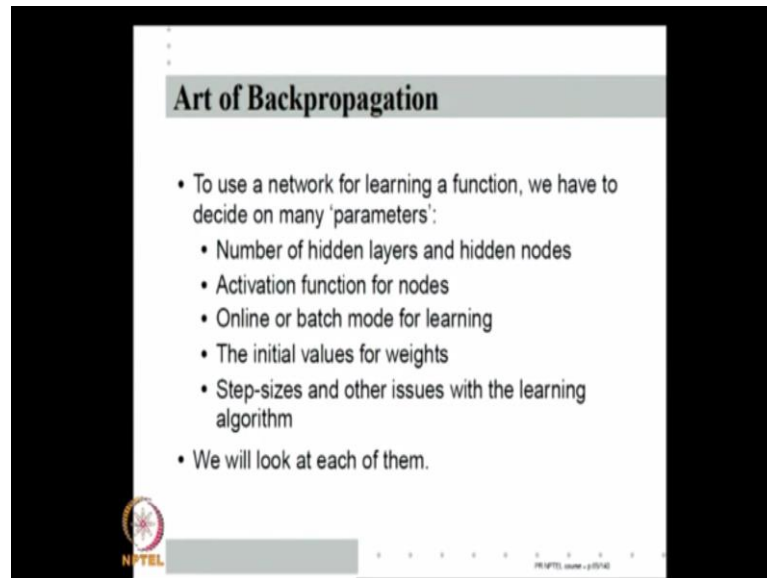


So, we almost done so we can say Neural network models are seen to be. We know how to train Neural networks for learning functions for learning classifiers we can learn non-linear regression function non-linear classifiers we can we are essentially doing empirical risk minimization error loss function and for minimization we are using gradient descent. The back propagation is a very elegant way of doing this so we model the algorithm everything is clear and while the course so we cannot see too many examples. Neural network models are seen to be quite good for classification regression that mean using many applications most of your math lab tool kits and so on.

Always contain this now and in many application that is into be quite effective but, to actually use them, now we are we are moving to random where models are complicated to actually use these models are there many parameters of the network that need to be chosen or chose the hidden layers number of hidden nodes step sizes in the learning algorithm activation function types there are lot of things to be chosen right and you know the gradient descent can get stuck in local minima. So, we want to get to better local minima. So, can I tweak my initial points are there. Some other things I can do for

the gradient descent, all these are important of course, there is very little theory in this but, we will from the next 2 slides. I will discuss a few practical tips of how to make learning with neural network more effective ok.

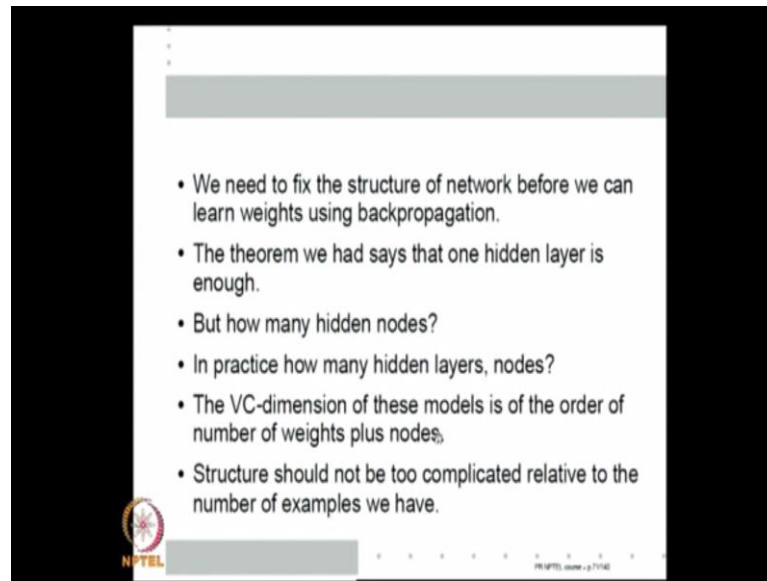
(Refer Slide Time: 31:02)



So, basically the issue is so that is why call this the Art of the Back propagation. The science is over now but, you know in practice it has to work well so use a network for learning functions, where we have to decide on many parameters right number of hidden layers and hidden nodes activation function for nodes other we introduce online or batch mode for learning. What the initial values for weights to use a learning algorithm as a iterative gradient descents?

So, we have to start somewhere and because the gradient descent converges to local minima close to the starting point initial values can make a lot of difference right how do I chose step sizes other issues learning algorithm to the first 2 are, how to fix the network structure last 3 are what I can do about the learning algorithm, this need to be discuss we need some to show all of them so we will going to look at each of them.

(Refer Slide Time: 32:01)



First about the structure before we can learn the weight would fix the structure the network right that we know back propagation only learns the weights after the structure the network is fix because we need to do the forward and backward computation should do back propagation so the structure the network has to be completely fix before I can do back propagation. Now we have seen the theorem that says one hidden layer is enough, if you have sufficient hidden nodes right. So, basically we can always chose one hidden layer, but we do know how many hidden nodes to use there is no theoretical thing about how many so the just do trial and error essentially.

On the other hand I do not want to use too many hidden nodes because I may over fit it. We will, I will come a little while later to what a over fitting is some time using 2 hidden layer is quite be better suppose I got a 50 dimensional feature vector. Let us say I used I want the 3 layer network with one hidden layer and lets a ten hidden nodes because input is 50 inputs layers 50 nodes I got 50 into ten 500 weights and then ten into one let say one output node 500 and plus ten weights. But suppose I put this ten hidden nodes not as in layer but, one layer of 5 and the next layer of 5 I use 2 hidden layer then input layer to the first hidden layer is only 500 into 5 is to 50 first hidden layer to second hidden layer is 5 into 525 and another 5 for second hidden layer to.

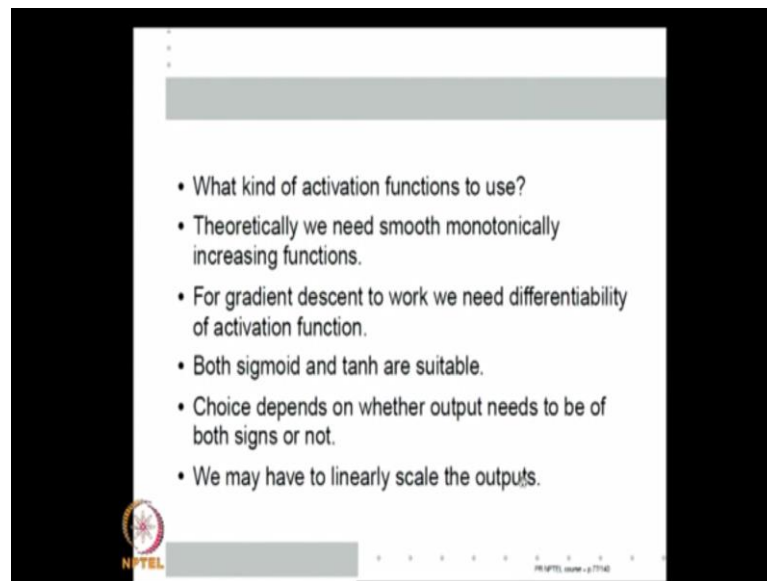
So, instead having you know 510 weights I am doing with 280 weights now whether 10 hidden nodes in one layer is equivalent to 5 each in 2 hidden layers. Nobody knows, but

very often this is one reason why I may use more than one hidden layer right so that I can reduce the number of weights. So, in practice one never uses very many hidden layers use you try one or 2 hidden layers not much more number of hidden nodes is always by trial and error. We do not want to put too many, you put just a few to see that we are getting good enough results, if you are not getting good enough results than you like to increase the nodes.

How do did I said how shoe is shoe a term of the VC-dimension of these models is of the same order of the number of weights plus number of nodes. So, we know that we need error management chose more than VC-dimension examples so depending on how many examples we have we would like to constrain a network structure where the often this network is not possible to have examples which are ten times the number of weights and so on. Because the weights are very many in this but, we still do not want to be too large at least you want a number of examples to be say of the same order as the weights we mean at least a little more than the weights. So that also determines how many hidden nodes will use we know how many training examples we can afford and that determines how complicated the structure can be.

So, this is both the only guide we have we know the VC-dimension is of the order of number of weights plus nodes and hence looking at the number of pairing samples. We have we would like to keep over total number of weights counts and we used that as a guide to decide how many hidden nodes we can afford so that is about all I can say about hidden nodes. Most of the time it is trial and error and a little later in the codes we look at what do you mean by trial and error we look a techniques called some of techniques just cross validation which will help you to try out different number of hidden nodes and say which is better ok.

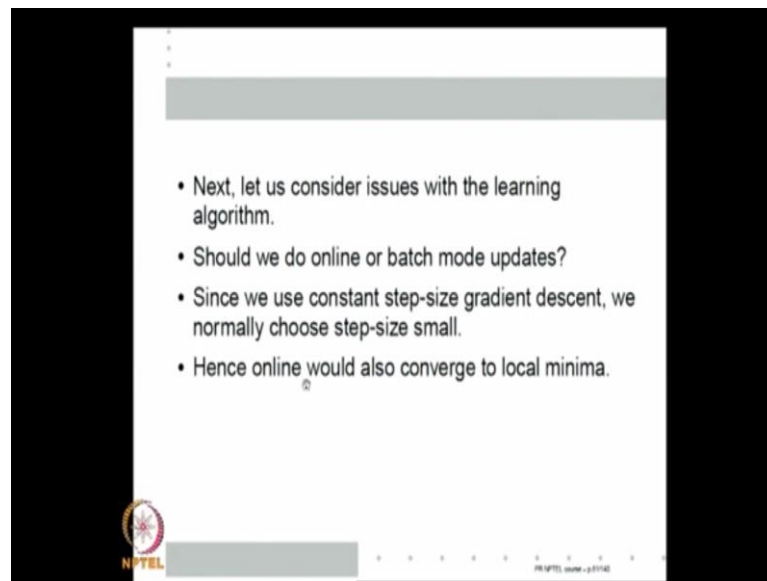
(Refer Slide Time: 35:55)



Now what about activation functions? Once again the theorem that we stored says that it should be smooth monotonically increasing functions of course, monotonically increasing or not we any way need smoothness because to do gradient descent we need differentiability of the activation function. So, that much is certainly true for using back propagation. Given this what can I use the most commonly used activation functions are sigmoid and tan hyperbolic tangent and sigmoid between them the choice depends on what kind of output you needs see sigmoid gives you output range only 0 to 1, where as tan hyperbolic gives minus one to one as we said we can always scaled the output. We can make the output node so actual output to be k times sigmoid or k time the tan hyperbolic so but, the question is whether I need output of only one sign or I need outputs of both signs right.

So, depending whether I need if I need outputs of both signs then I have to use a tan hyperbolic kind of function otherwise I use sigmoid these 2 of the most often used function then many other functions one can use. For example, it is power of minus not input square that is also a nice function dictating the function so there are other kinds of activation functions that can be used in practice. But, most often in a vast majority of applications people use either sigmoid or tan hyperbolic activation functions as I said we have to always linearly scale the output to match the actual output range right inputs we will come to the inputs as later but, outputs if the output range is beyond 0 to one or beyond minus one to one then we have to linearly scale the outputs.

(Refer Slide Time: 37:54)



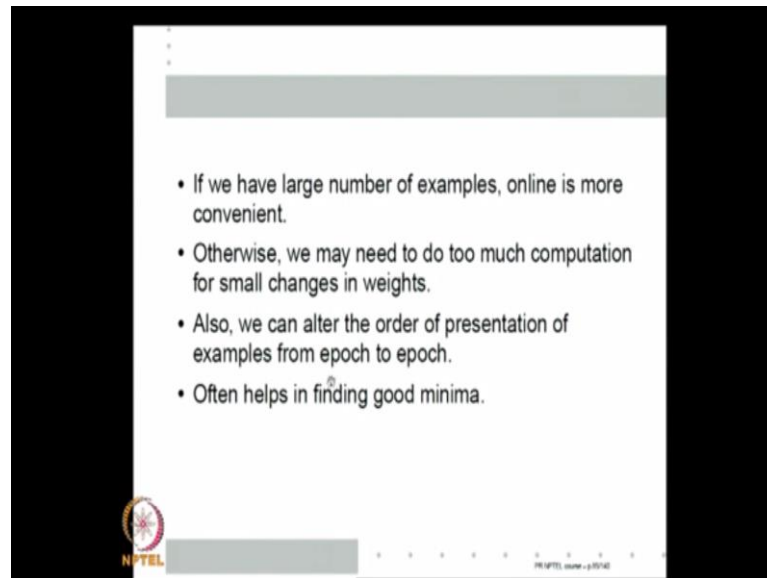
Next, let us consider all the issues that deal with the learning algorithm why the issues in learning algorithm. First should I do online or batch just to recall for each training example I put that input at the input at the network calculate the output. Once I have the output I used the desired output for that input to calculate the error for that error I can calculate all the gradient descent all the partial derivatives in back propagation.

Now I have to do this for everyone of the inputs in the batch mode update I keep the weights all fixed apply inputs one by one at the input under the network calculate the outputs errors and by through back propagation all the partial derivatives and then do one gradient descent step one update of the weights in the online mode. I put one example calculate the output immediate the error back propagation find all the partial derivatives and immediately update the weights and then apply the next example of course, when I do this in the online mode I can apply the examples in some order so going through all the examples once is called an a pock normally in the online model.

Now, most often a theoretically it may not make much difference as I mentioned when we did the linear models and when we first look at the online and batch things the l m s algorithm. If the step size is sufficiently small then both online and batch updates have the same asymptotic behavior. Now in this particular case when you using back propagation we are doing gradient descent on a very high dimensional and very highly non-linear objective function and we are using a constant step size gradient descent. So

we have to any way use a very small step size where you any way use a very small step size you know both of them will have similar asymptotic behavior and so both of them will converge to local minima. So the choice is not so much about the earlier converge.

(Refer Slide Time: 40:17)

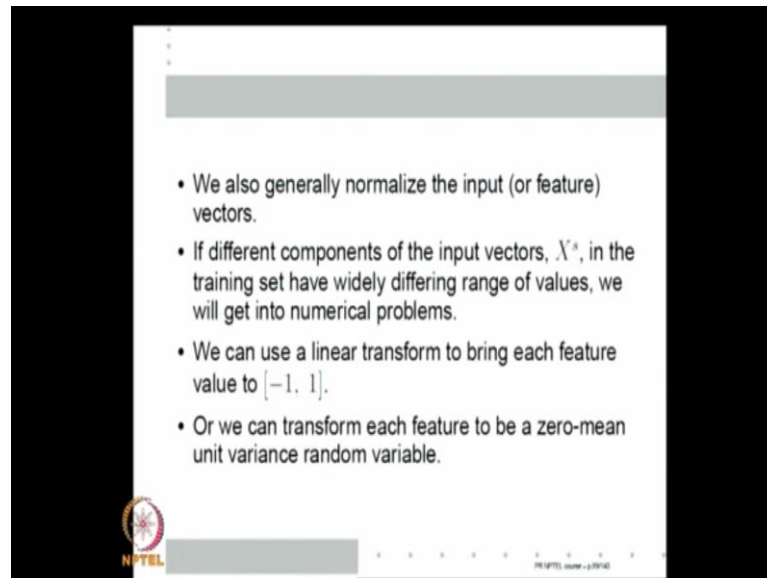


What may be more convenient implement if we have large number of examples online is more convenient of course, batch mode is nice because batch mode is exactly the same as the right gradient descent of the function. So, I can actually update batch mode but, if I have large number of examples online might be better because otherwise you know we may have to run thorough some thousand of examples so we have to do thousand function computation and then only my weights will get updated by small bit I have to any way use a small step size. I cannot afford to use very large step size so even to change my weights by small amount. I may have to do lot of computation calculating outputs for you know 1000 input patterns. So, might be wasting to much computation. So, this is the reason why if you have large number of examples very often online is done.

Another is I can, when a 2 online unlike batch I can possibly use some no some kind of randomization techniques to try to get to better local minima this is gradient descent. So, we need all kinds of tricks to get good minima so for example, from epoch to epoch I do not have to present all my examples in this same model. So, may be in one epoch of the online algorithm I present examples one to n and the next epoch I go from n to one or I

do a random shuffling of the examples for each epoch very often that kind of things help in finding good minima so it kinds of gives an extra shade to the algorithm so this tricks generally find you good minima.


(Refer Slide Time: 41:51)



Another thing that is often needed is to normalize the inputs or the feature vectors why if different components of the input vector X have different very widely differing range of values. So if one component goes from point one to point 2 another component goes from one to ten thousand then ultimately you know you take the inputs multiply by weight and add them right I get into all kinds of numerical problems right even you know my weights have to traverse very large dynamical range and that but, be very difficult for the gradient descent algorithms to find. So, because of that I would like to have all the components in put to roughly as a same dynamic range when we are going it is to use a linearly transform every range into minus 1 to 1.

So, simplifying the max and min for each of the input dimensions and then simply linearly transform that input so that all components will be minus 1 to 1 but, what often gives better performance than this is to transform this features. So, that each feature that is each component of x can be thought of as a 0 mean unit variance random variable right so that I am not really making a strictly in minus 1 to 1 but, I am transform them so each of them can be would be a 0 mean unit variance random variable.

(Refer Slide Time: 43:21)




- Let $X = (x_1, \dots, x_m)^T$ be the feature vector.
- The training examples,
 $X^s = (x_1^s, \dots, x_m^s)^T$, $s = 1, \dots, N$,
can be taken to be *iid*.
- We can estimate mean, μ_j and variance, σ_j^2 of j^{th} feature as

$$\mu_j = \frac{1}{N} \sum_{s=1}^N x_j^s \quad \text{and} \quad \sigma_j^2 = \frac{1}{N} \sum_{s=1}^N (x_j^s - \mu_j)^2$$

PR-14773, slide - 9 (37/40)

How do you do that let say this is the feature vector with m components. Now, I have training samples X^s that mean I have a sample for each one of this components N such training sample these can be taken to be iid so, if I think of x_1 at the random variable correspond with the first feature x_1^s s is equal to 1 to N on iid samples of that random variables so I can find the mean N variance. So, the mean μ_j and variance σ_j^2 of the jth feature as these are the maximum likelihood estimates. In general both of them are sample mean and sample variance these are very good estimates for most kind of models. So, i can estimate my mean and variance of the of each of the feature components.

(Refer Slide Time: 44:14)



• Now we can transform each example X^s to $\tilde{X}^s = (\tilde{x}_1^s, \dots, \tilde{x}_m^s)^T$ by

$$\tilde{x}_j^s = \frac{x_j^s - \mu_j}{\sigma_j}$$

• This will make each component of \tilde{X} , to be a zero-mean unit variance random variable.

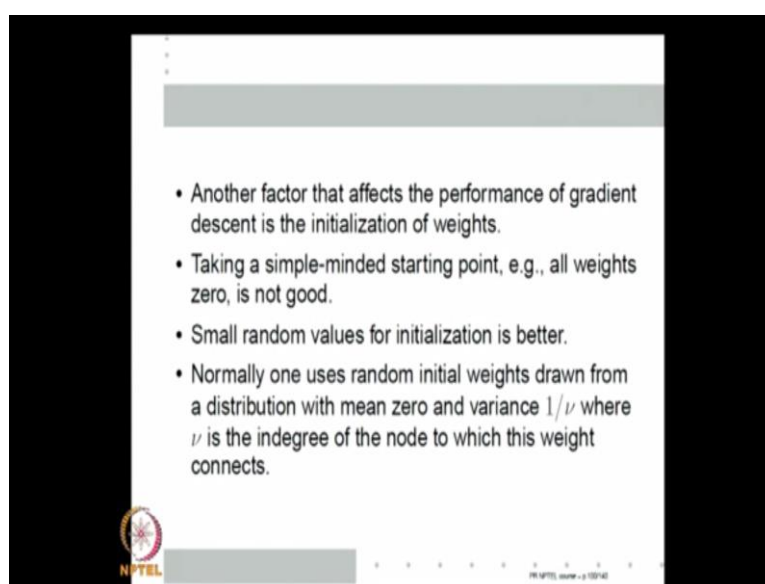
• This is often better than simply transforming the range of each feature component into $[-1, 1]$.

• We can also use some linear transform to decorrelate the feature components.

Once I do that I can transform the original example X^s to a new example \tilde{x}^s so that each component now become \tilde{x}_j^s so the \tilde{x}_j^s component x_j^s as become x_j^s minus μ_j by σ_j σ_j is of course, square root of σ_j^2 . So as you know the because x_j^s as mean μ_j on variance σ_j^2 \tilde{x}_j^s will I mean 0 variance one right so this transformation will kind of make all the feature vectors to be 0-mean unit variance random variables. And often this kind of transformation works better than simply doing a linear transform to bring all feature components to minus 1 to 1 because here by this one by σ_j I am kind of also normalize the variation in each of this random variables a matter of fact here. Of course, I am only individually making them 0 mean unit variance I am not worrying I am not doing anything about correlations between them.

Similarly, linear transformation is possible to do so that different feature components can be uncorrelated or decorrelated. So, that you know learning can be much more effective if you have independent features we will look at such transforms when we do what is called p c a principal computer analysis. So, this is about normalizing inputs right so I have to normalize inputs.

(Refer Slide Time: 45:40)

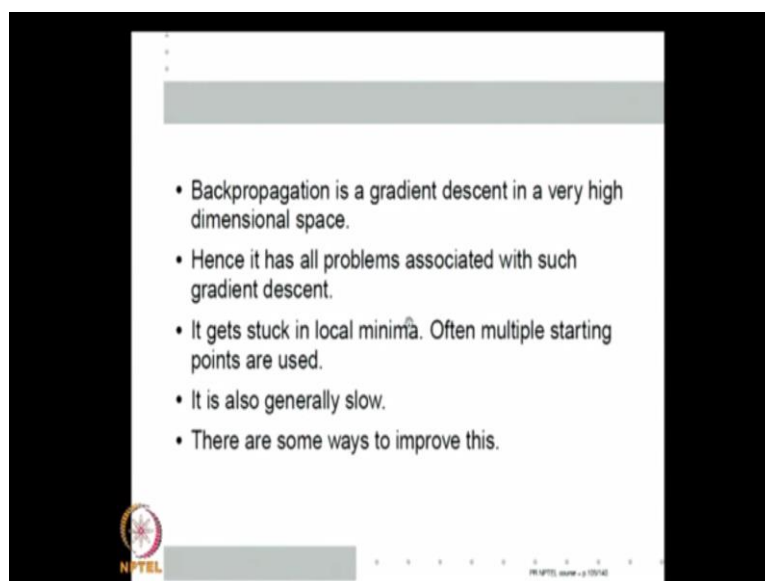


Another factor that effects performance of any gradient descent is where you start right generally it converges local number of closest to where you start. So where you start becomes often very crucial and because the very high dimensional space a simple-minded starting point like take all weights to be 0 is never a good starting point see you do but, that also truly there is lot of weight space cemeteries if I permute the nodes in the hidden layer. So, I get a different weight vector but, which effect will represent the same function which means you know the minimizer minimizing weight vector will not be unit there will be many weight vectors where minimum can be achieved and given so much.

So many such weights pits cemeteries and the generasy all 0 kind of initial point is generally not a good initial point since has not much to say except that often small random values for initialization is better. Instead of all being same some initial random variation will allow the gradient descent search to be much more effective normally what one does is to chose the initial weights from some distribution chose them randomly where the distribution as 0 mean and variance one-mu th when mu is the indegree.

So depending up on how many weights are there for each node or for all the weights of one node I use one distribution this one-mu th variance is once again it just a guidance. For example I can decide on the range of weights I can think I am to be uniform from minus a 2 a where a depends on the indegree right but, basically the idea is that for nodes that have large m number of weights I do not want to much variation.

(Refer Slide Time: 47:58)



• Backpropagation is a gradient descent in a very high dimensional space.

• Hence it has all problems associated with such gradient descent.

• It gets stuck in local minima. Often multiple starting points are used.

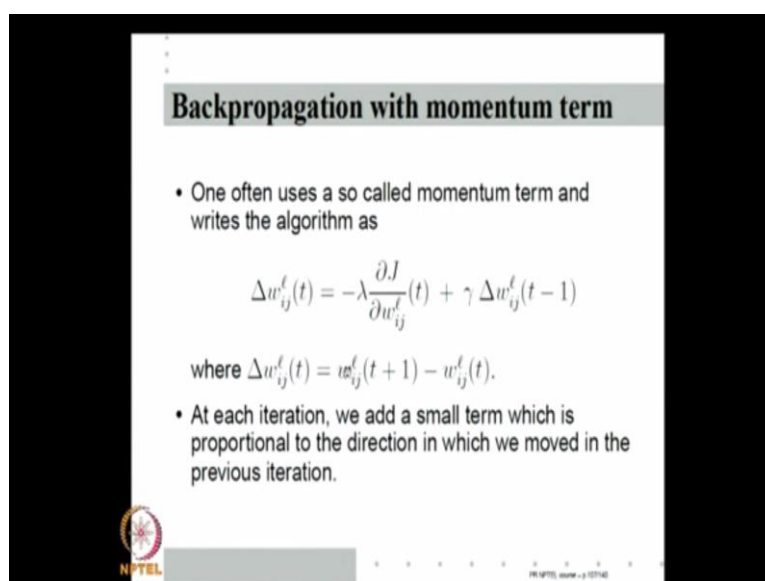
• It is also generally slow.

• There are some ways to improve this.

NPTEL

Yeah essentially back propagation is a gradient descent in a very high dimensional space so a gradient descent kind of optimization in high dimensional space has lot of problems. So back propagation will have all the problems associated with any such gradient descent algorithm right. For example, it gets stuck in local minima so very often what people do is used multiple starting points so run it from many different starting point and try to take the best one you can right this is how this is also done and we know it get stuck in local minima is very slow gradient descent is very slow and so on there a couple of things that we can do to improve the algorithm right improve the gradient descent itself.

(Refer Slide Time: 48:46)



Backpropagation with momentum term

• One often uses a so called momentum term and writes the algorithm as

$$\Delta w_{ij}^{\ell}(t) = -\lambda \frac{\partial J}{\partial w_{ij}^{\ell}}(t) + \gamma \Delta w_{ij}^{\ell}(t-1)$$

where $\Delta w_{ij}^{\ell}(t) = w_{ij}^{\ell}(t+1) - w_{ij}^{\ell}(t)$.

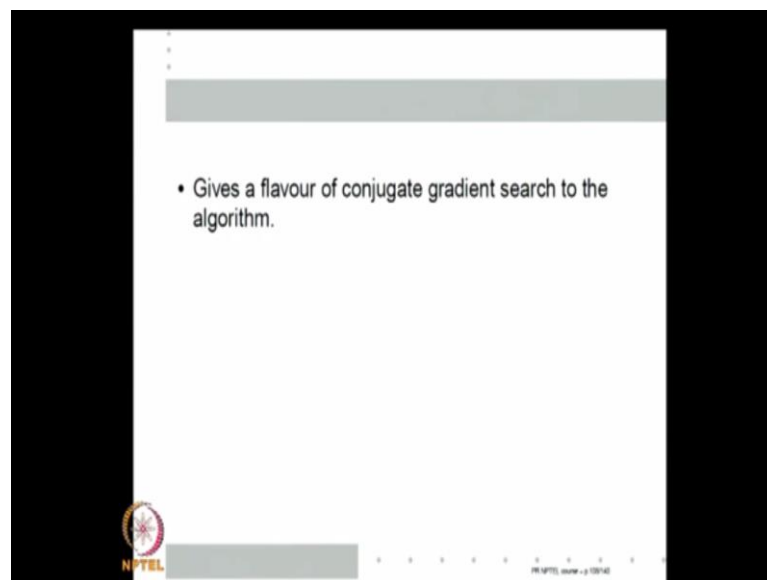
• At each iteration, we add a small term which is proportional to the direction in which we moved in the previous iteration.

NPTEL

Here are a few such things this is called Back propagation with a momentum term what you do is if I take if I write Δw_{ijl}^t as $w_{ijl}^{t+1} - w_{ijl}^t$ essentially what the algorithm we had earlier $w_{ijl}^{t+1} = w_{ijl}^t - \text{the gradient right} - \lambda \text{ time the gradient}$. So, if I take this difference to be Δw_{ijl}^t so my earlier algorithm is Δw_{ijl}^t is simply minus the gradient is equal to minus $\lambda \frac{\partial J}{\partial w_{ijl}^t}$ this simply mean that this gradient is evaluated meeting the values of weight set t .

So, what I am doing in this algorithm is in addition to the usual gradient step I am adding one more step what is the second step give me this is Δw_{ijl}^t minus one that is $w_{ijl}^{t-1} - w_{ijl}^t + 1$. So, essentially in addition to the current gradient I am adding a term which is proportional to the direction in which we moved in the previous iteration if you think of it as vector this is the component equation look at the vector equation then in addition to the gradient vector this one adds a component which is the direction in which I traveled traverse in the previous 2 iteration. So, I am currently t to $t + 1$ iteration this is the direction which I traveled between $t - 1$ to t iteration right so at each iteration where a small term which is proportional to the direction in which we moved in the previous iteration in addition to the gradient.

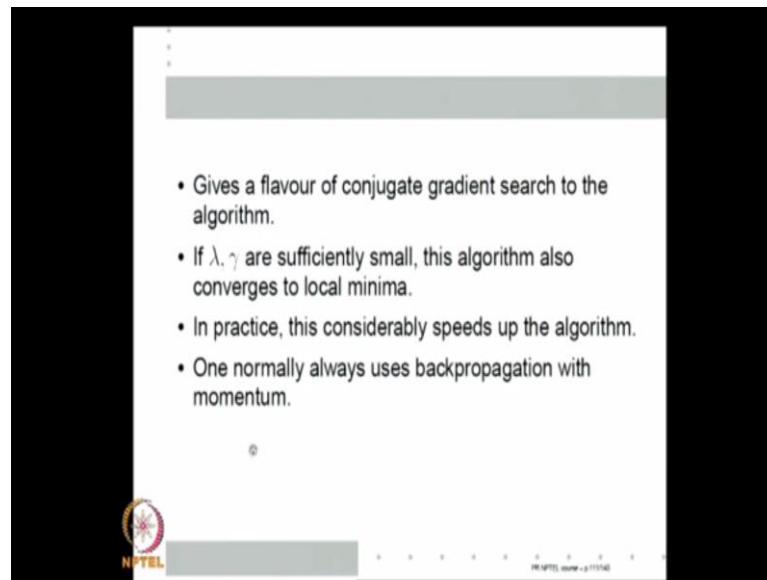
(Refer Slide Time: 50:35)



What it that do this essentially is what a conjugate gradient descent algorithm of course, the proper conjugate gradient needs to correctly find previous direction it has to be the


weight for that is to be determined based on the proper conjugate directions but, essentially can think of this as the poor man's conjugate directions method. So, this essentially adds a kind of conjugate directions kind of flyover to the gradient descent and hence peace setter.

(Refer Slide Time: 51:23)



So, now we have 2 step sizes a λ the step size for the gradient term and the γ as a step size for this term this term is often called the momentum term so if both the step sizes of the gradient and the momentum term are sufficiently small then this algorithm also be shown to converge to local minima. So, essentially it does not change the asymptotic properties whether that term is there or not I still go to the local minima but, in practice this term considerably speeds up the algorithm this is just like gradient descent and conjugate direction by conjugate directions method is faster even though it also converges to the same minima ok. So, in practice one always uses back propagation with a momentum term no body our uses it without the momentum term.

(Refer Slide Time: 52:02)




- Another simple strategy that one uses in the learning algorithm is the so called weight decay.
- From time to time we replace each weight w_{ij}^t with $w_{ij}^t(1 - \epsilon)$ where ϵ is a small positive number.
- We can do this every iteration, or after a fixed number of iterations and so on.
- This is essentially implementing gradient descent on a 'regularized' risk.

NPTEL course - CS111140

Another simple strategy that one often uses for learning is what is called weight decay what do you mean by weight decay you say from time to time we replace each weight with one minus epsilon times itself where epsilon is some positive number. So, I just shut a part of the weight the idea is that this way weight do not blow up I do not let weights become very large ok. We can do it in every iteration we can do after fixed number of iterations and so on where is this work this essentially implementing gradient decent on a regularized risk we seen regularized least squares last time.

(Refer Slide Time: 52:35)



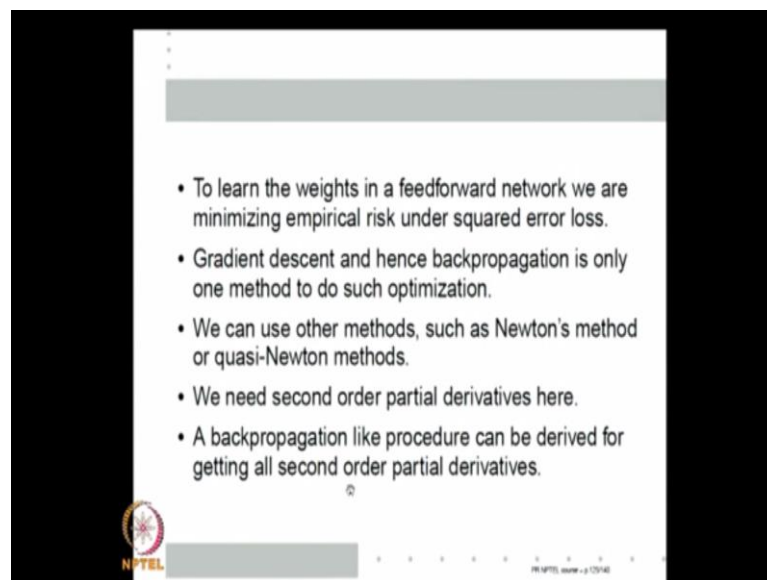
- We are trying to minimize J , sum of squared errors.
- Suppose we decide to minimize
$$\tilde{J} = J + 0.5 \epsilon ||W||^2$$
- This is the regularized least squares that we discussed earlier.
- Now gradient descent would give us the algorithm
$$w_{ij}^t(t+1) = w_{ij}^t(t) - \lambda \frac{\partial J}{\partial w_{ij}^t} = w_{ij}^t(t) - \epsilon w_{ij}^t(t)$$
- This corresponds to the so called weight decay.

NPTEL course - CS111140

So, what is that we essentially trying to minimize J the sum of squared errors suppose instead of J we want to minimize \tilde{J} where $\tilde{J} = J + \frac{\lambda}{2} \|W\|^2$ so this is exactly like the regularized least square that we discussed earlier. Instead just using the data error we also use a model complex term and for squared error we seen that a good complexity term is square of the weights the norm of the weights.

So, what is that mean now if I do gradient descent and \tilde{J} now this will be the usual gradient minus λJ this and this will give me minus epsilon the 0.5 I have put here so that this two will cancel this is nothing but, the square of each of the components. So if I take a partial derivatives of any one component I get that component vector so making each w_{ij} one match epsilon time that w_{ij} from time to time is simply that on top of the gradient descent I am just implementing the regularized least squares so essentially where decay is same as doing a regularized least square gradient descent is that doing gradient descent are not just the sum of squares of error but, sum of errors plus a regularization term where epsilon now plays a roll of the regularization constant ok. so these are all the various tricks that one can use to make the gradient descent work better.

(Refer Slide Time: 54:07)

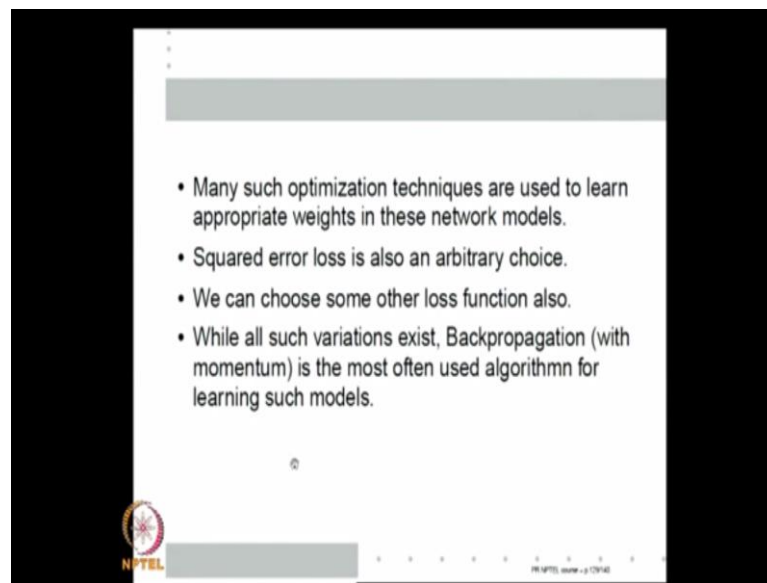


So, to sum up to learn weights a feedforward network we are minimizing empirical risk under squared error loss and gradient descent and hence back propagation is only one method to do such optimization right. Let us not forget that what we want to do is minimize empirical risk under squared error loss and to minimize we decide to use \tilde{J}

iterative gradient descent and a calculate gradient were using back propagation. They can be other methods for the same objective function I can use many other methods for minimizing for example, I can use Newton's method I can do any of the quasi-Newton methods such as Powell method or b f g s and so on right.

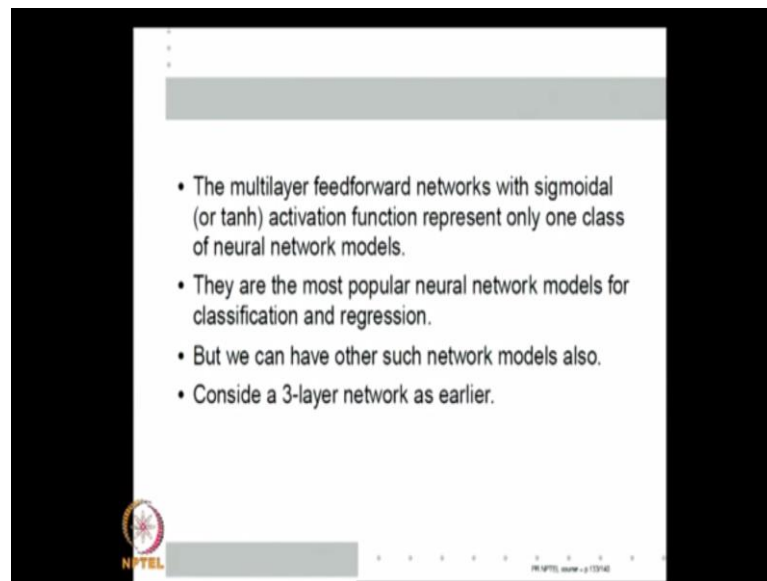
There many Newton or quasi-Newton methods all of them need second order partial derivatives some time they need to inversion sometimes quasi-Newton methods do not want to the inversion but, all of them need the second order partial derivatives. As it turns out if a back propagation like method exist of first order derivatives a similar procedure should exist for second order derivatives even though is a little more complicated. It is possible to get the second order partial derivatives also using the back propagation like method. So, I can use such quasi-Newton methods is not all.

(Refer Slide Time: 55:21)



So, we can use many optimization techniques to learn the appropriate weights because squared error loss itself is an arbitrary choice. I can use other loss functions right so the many variations are possible within this networks I many other variations to learn within this network while all such variations exist. And people have used other optimization techniques also rarely some other loss function in most often use algorithm is back propagation with this models.

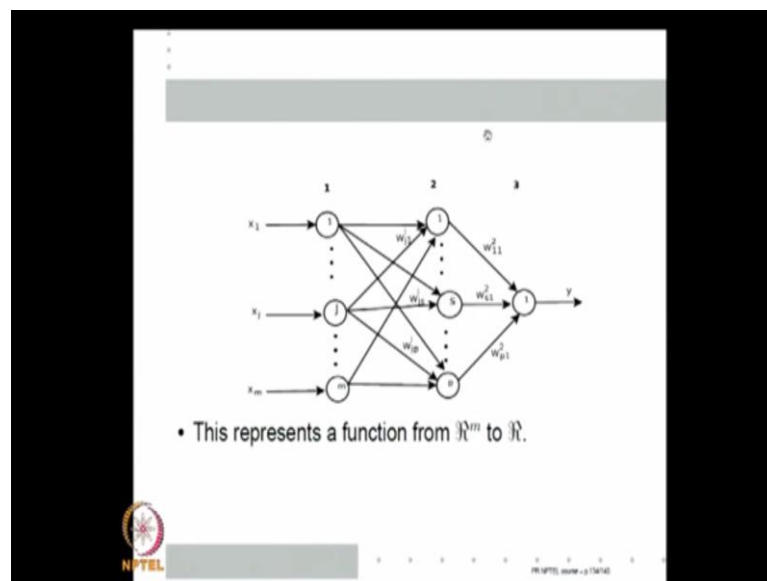
(Refer Slide Time: 55:52)



- The multilayer feedforward networks with sigmoidal (or tanh) activation function represent only one class of neural network models.
- They are the most popular neural network models for classification and regression.
- But we can have other such network models also.
- Consider a 3-layer network as earlier.

Now, this completes the multilayer feedforward network models with sigmoidal activation functions where each node takes a linear combination and passes it through a sigmoidal function, but these are not the only kind of neural network models one can have. They can be other network models while these are the most popular neural network models for classification and regression. One can have other such network models. Let's just take a look at what I mean by that.

(Refer Slide Time: 56:20)



The diagram illustrates a 3-layer neural network with three layers labeled 1, 2, and 3. Layer 1 (input) has nodes x_1, x_2, \dots, x_m . Layer 2 (hidden) has nodes $1, 2, \dots, n$. Layer 3 (output) has node 1 . Weights are labeled w_{ij}^1 between layers 1 and 2, w_{ij}^2 between layers 2 and 3, and w_{p1} for the output node. A bullet point states: "This represents a function from \mathbb{R}^m to \mathbb{R} ."

Let us go back to the old 3-layer network essentially the 3-layer network function

depends on its a linear sum of the outputs here so given the inputs I first calculate these outputs to any other function and then take a linear sum right.

(Refer Slide Time: 56:36)

- The function represented by this (with linear output node) is

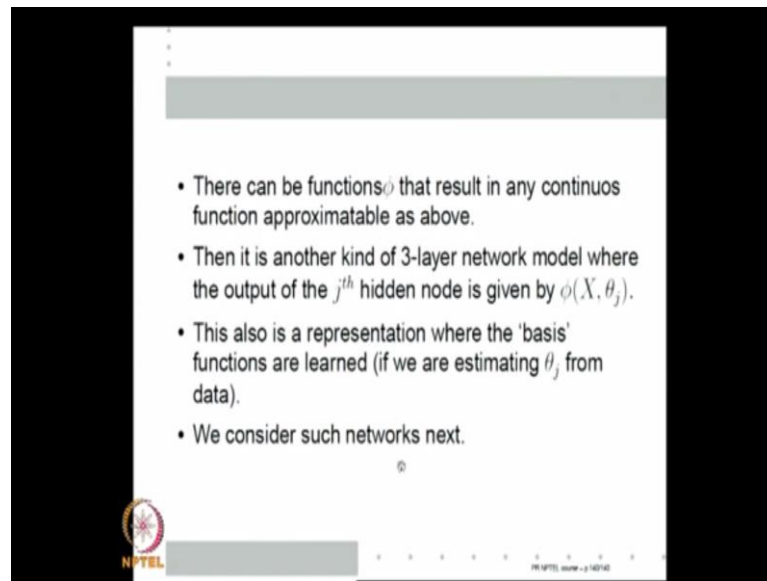
$$y = \sum_{j=1}^p \beta_j f\left(\sum_{i=1}^m w_{ij} x_i + b_j\right)$$

- We can rewrite this in a form

$$y = \sum_{j=1}^p \beta_j \phi(X, \theta_j)$$

So, the function represented by the network can be witnessed y is equal to by sum of j the hidden nodes β_j into sum this is the actual sigma (()) function and each node does linear sum. I can rewrite this function as this f of all this can witness some ϕ of X the input vector comma θ_j by θ_j (()) all $w_{ij} b_j$ everything. So this is some function ϕ which is a function of x and some parameters and the parameters are specific to each of the each of the hidden nodes each of the j right.

(Refer Slide Time: 57:13)



So, there can be functions ϕ that result in continuous function approximation they can be weight of representing any continuous function like this also because this is just like this. This is also you know because the $\phi(x, \theta_j)$ but, where θ_j have I have show able parameters here so it is not a fixed basis of expansion. So this also a representation where the basis functions are learned if we are learning θ_j from and I can think of the this is another kind of a 3 layer network model where I can still think of this is a 3- layer network where $\phi(X, \theta_j)$ is the output of the j th node and then multiply with β_j and sum it up right. So I can have a different 3-layer network model whose output is like this and this is also not a fixed basis function but, basis function adopted so these are the kind of networks these are called you know perfect interpolation networks radial basis function networks and so on these are the network models that we will consider next is in next class.

Thank you.