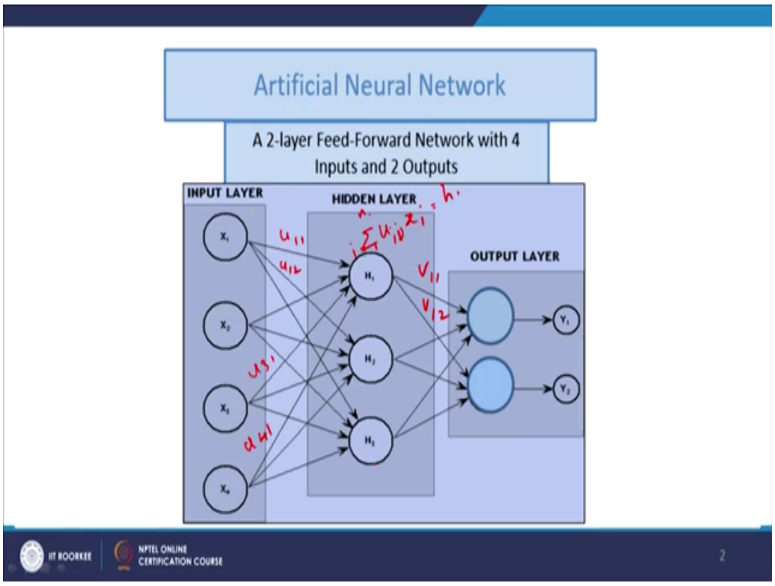
Robotics and Control: Theory and Practice

Prof. N. Sukavanam
Department of Mathematics
Indian Institute of Technology, Roorkee

Lecture - 19
Artificial Neural Network

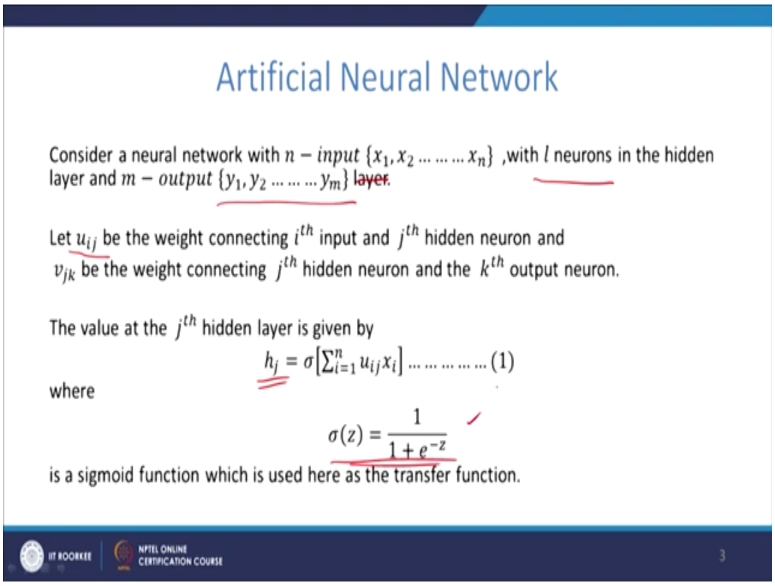
In this lecture, we shall see how neural networks can be used to approximate air function. Also we shall see how a neural network can be used to solve inverse kinematics of a robot manipulator and how to generate a trajectory given initial and final conditions. So, Artificial Neural Network is depicted in this picture.

(Refer Slide Time: 00:53)



So, it has 3 layers one is the input layer, the middle one is the hidden layer and the last one is the output layer. So, for example, here x 1 x 2 x 3 x 4, these are the inputs and y 1 y 2 are the outputs, 4 inputs and 2 outputs are given in this artificial neural network and there are 3 hidden neurons.

(Refer Slide Time: 01:19)

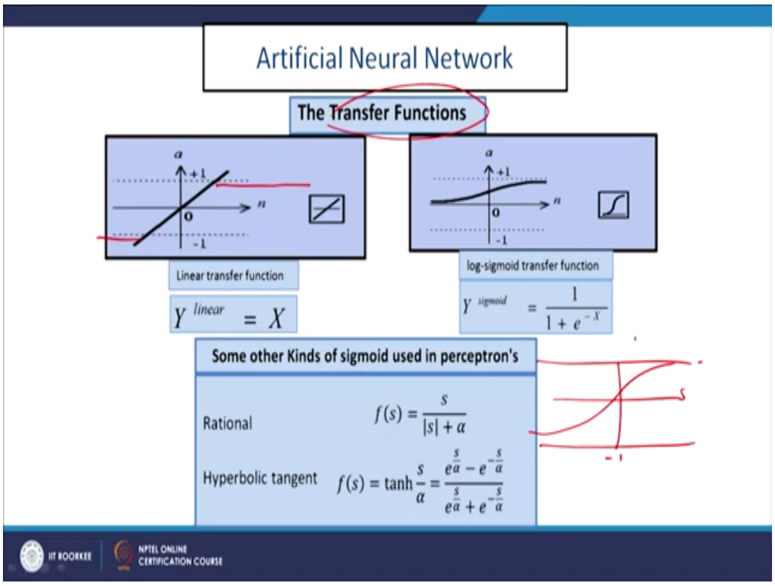


So, here in general let us consider x 1 x 2 x n n inputs and there are l hidden neurons and y 1 y 2 y m are m outputs. So, the input and the hidden layer are connected by the weights given by u i j. So, in this picture; we can see that the first input x 1 is connected with the first hidden layer by the weight say u 1 1 and with the second hidden neurons it is u 1 2 the weight is given by this. And the forth input with the first hidden layer we say that it is u 4 1.

So, similarly the hidden layer first hidden layer with the first output is connected by the weight v 1 1 and then it is v 1 2 with the second output. So, in this manner they are connected and they are the calculation, the value of the hidden neurons j th hidden neurons is calculated by the function sigmoid function, sigma of the summation u i j x i. So, that is the value of the h 1.

So, for example, we multiply x 1 with u 1 1 and x 2 with u 2 1 x 3 with u 3 1 etcetera. So, the summation of u i j x j xi i is equal to 1 to n. So, this gives the summation the value h j h 1. So, here it is its 1 u i 1. So, this value is h 1 similarly h 2 h 3 can be calculated, then this its a real value and it is operated with the function sigma where the sigma function is given by this expression. Sigma of any real number, z is 1 by 1 plus e to the power minus z is the value. So, this is one type of sigmoid function, the transfer function.

(Refer Slide Time: 04:11)

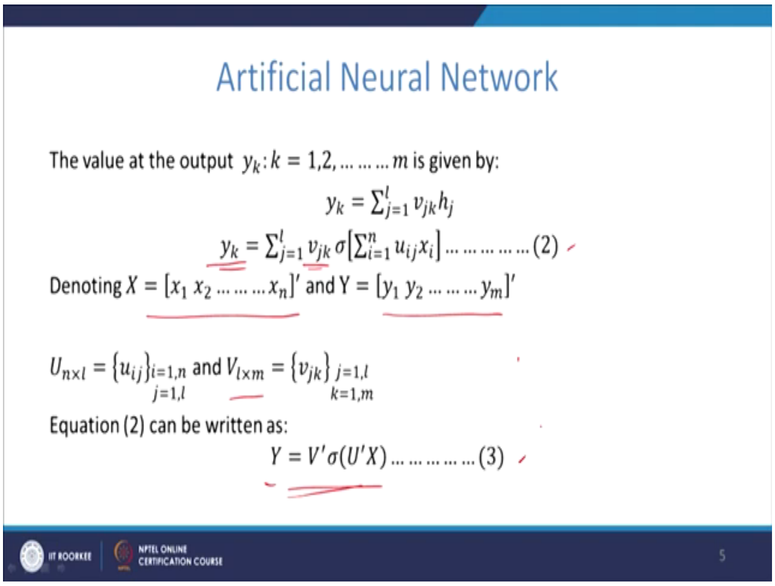


But, there are several types of transfer function which can be used in a artificial neural network. For example, y is equal to x it is an increasing function the value is between minus 1 and plus 1 in this region. And it is constant else where we consider the interval from minus 1 to plus 1 and the remaining values are constant. Similarly, the function 1 by 1 plus e to the power minus z; the graph of the function can be easily seen that it varies from the value 0 to 1 it is an increasing function. And, it is when it is when x is infinity it is the value is 1 and when x is 0 the value is half and when x is minus infinity the value is 0.

So, it is a increasing function and between 0 and 1 other functions are given by f of s is given by s by mod s plus a where a is any constant. So, here also the graph of this function is varying from minus 1 to plus 1 and it will look like this. When s is 0 the value is 0 and for infinity it is 1 and when s is minus infinity it is minus 1. So, similarly the tan hyperbolic function also will look like this. So, the property of any transfer function is simply it is an increasing function and it is bounded by 2 values either minus 1 to 1 or 0 to 1.

So, this way that is the property of a transfer function and one such example is this transfer function; sigma is z 1 by 1 plus e power minus z. So, the value of the hidden neurons is calculated by this expression.

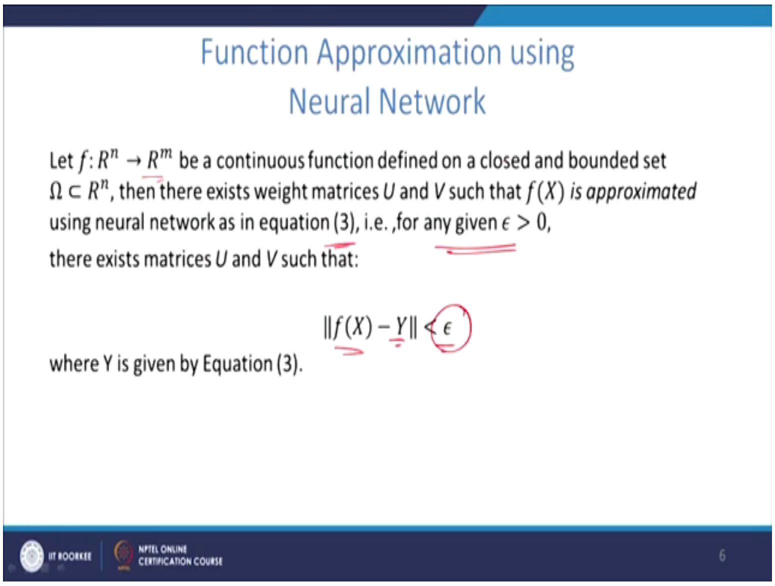
(Refer Slide Time: 06:22)



Then we get the output, the kth output y k is calculated by multiplying with the weight v j k of the h j value. So, j varies from 1 to l of v j k into sigma h j is given by this expression. So, for a given input x 1 x 2 x n we get the corresponding output y 1 y 2 y m using this artificial neural network.

So, instead of writing using this summation we can write them in the matrix form like this; if capital U is the matrix u i j. Capital V is the matrix v j k, the elements are given by this thing and l rows and m columns as given here. Then the expression capital Y, capital Y means y 1 y 2 y m transpose the column vector is calculated by V transpose sigma of U transpose X where X is this vector. So, the same expression 2 in the matrix notation is given in the form 3 for simplicity.

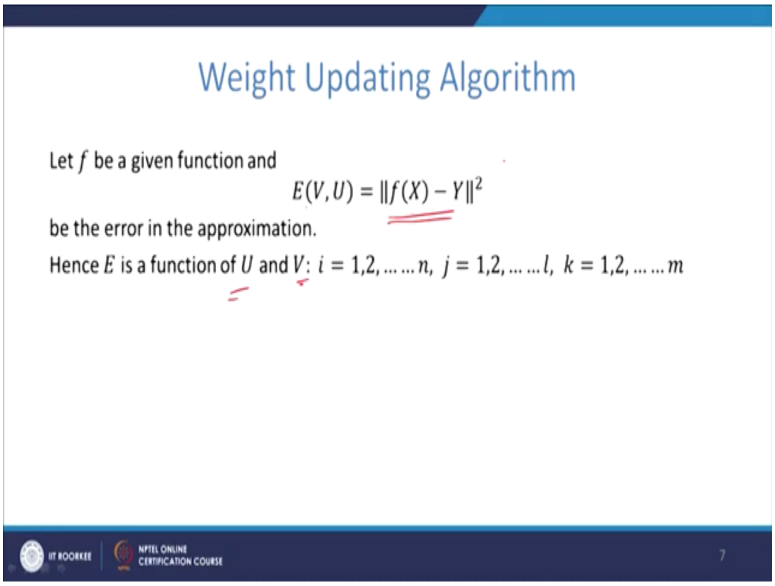
(Refer Slide Time: 07:43)



Now, the artificial neural network has a interesting property, that any given function f of n variables from R n and its value is in R m, it can be approximated by a neural network as given in the equation 3 in the previous slide. So, we can find suitable weights, weight matrices capital U and capital V in such a way that the function f of x and the neural network, the norm of the difference between f x and y is less than epsilon for any given epsilon.

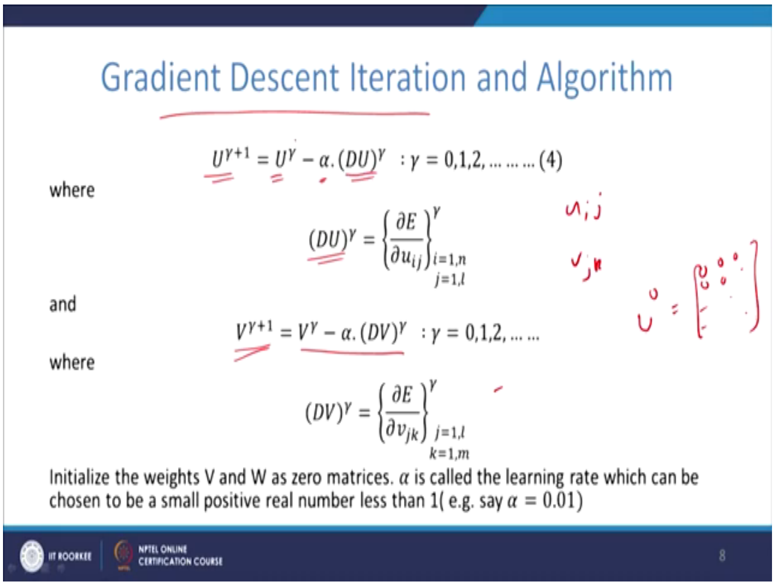
So, the property is like this. For if, f of X is a given function of n variable then we can find a neural network in the form of Y has given here in the equation 3 or in the equation 2 such that; the difference between the function f of X and the neural network is as small as possible. Whatever be the given epsilon; we can find the weight matrices U and V. So, this is a very nice approximation property of a any given function.

(Refer Slide Time: 09:09)



So, to find the weights what we can do is we take this as the error the difference f of X minus Y norm square is the error defined which is a function of U and V. Now, by minimizing this error we get the values of the matrices U and V. Once we obtain the matrices U and V by minimizing this error, we can substitute in the equation 3 in this expression which gives the approximate value of the function f of X.

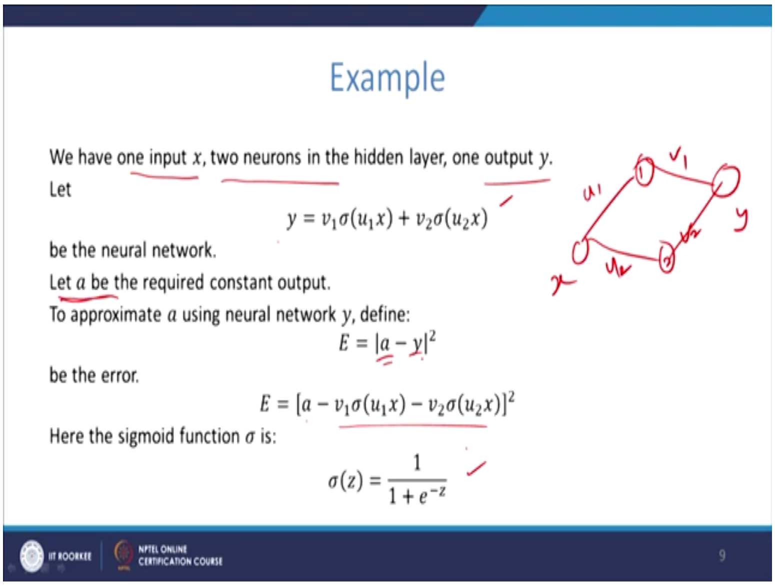
(Refer Slide Time: 09:49)



So, to minimize the error; we use the gradient descent iteration as follows the r plus 1 th iteration is depending on the r th iteration minus some small value which is called the learning rate alpha multiplied by the gradient of the expression U. Where D U is defined by this expression the rth stage, rth iteration the D U is defined by the partial derivative del E by del u i j. i varies from this thing because, e is the error which is a function of U and V where U and V are having the elements u i j and v j k as shown in the previous slide.

So, E is a function of all this u i j and v j k and the partial derivatives. These are all the matrices D U is also a matrix and capital E is a matrix and its derivative. Derivative of a matrix means; we have to differentiate at each element ok. Similarly V r plus 1 the value of V at the r plus 1th iteration is given by this formula, similar to this thing. Initializing the weights that is U 0 at the initial this thing we take the matrix 0 matrix only as the initialization in this algorithm then from that we can get 1 by 1 and finally, as r tends to infinity this will converge to a suitable weight which is required.

(Refer Slide Time: 12:02)

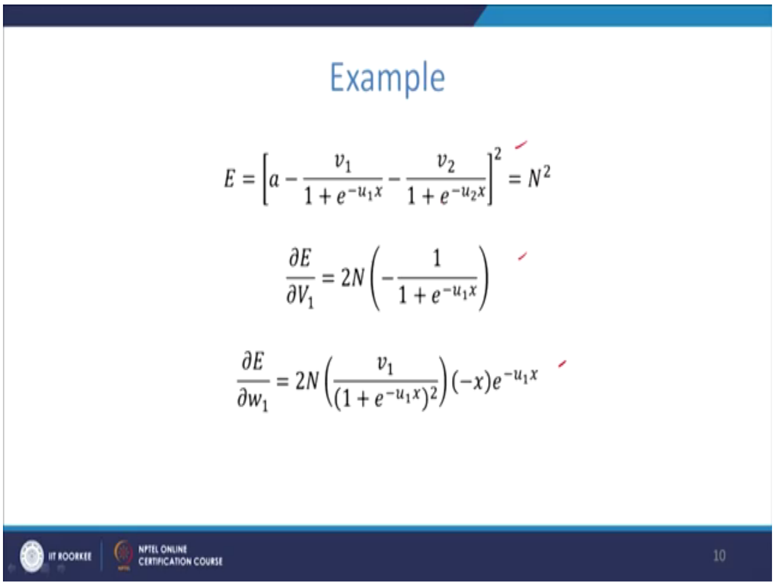


So, that is the standard procedure of the artificial neural network. So, let us consider a simple problem that is one input x and two neurons and only one output y. So, y can be calculated by this thing one input and connected by two neurons connected by one output. x is input y is output and this is hidden layer 1 and 2. So, we have the weight here u 1 and u 2, this is v 1 and v 2. So, v 1 into sigma of u 1 x plus v 2 into sigma of u 2 x is the calculation of the neural network.

And, if we want to approximate this constant a using a neural network, the error is the constant minus the neural network y as given here the square of the error to minimize that.

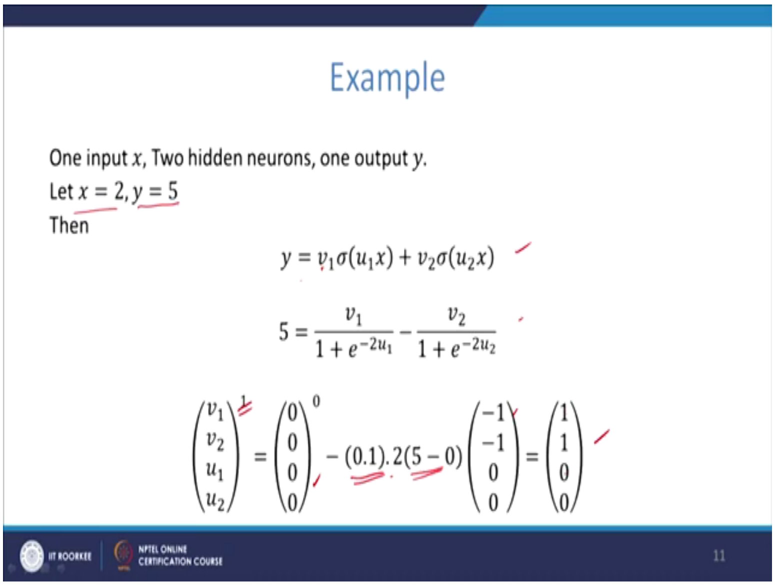
So, when we substitute y we will get E equal to a minus this whole square and sigma of z is taken as the transfer function, the sigmoid function.

(Refer Slide Time: 13:19)



And so error is written in terms of the sigmoid function like this; del E by del V 1 del E by del w 1 is like this. The partial derivative of this expression E where n is nothing but this bracket, square bracket expression. So, now similarly del V del E by del V 2 and del E by del w 2 can be calculated in the same manner.

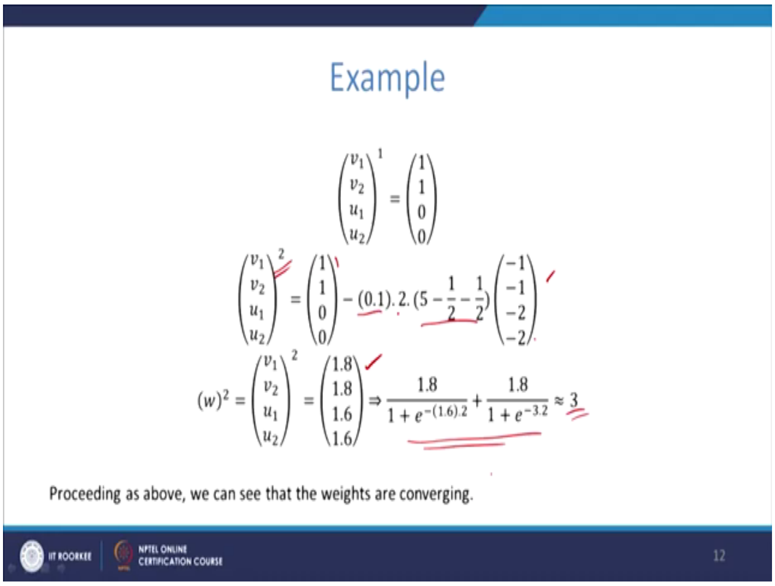
(Refer Slide Time: 13:51)



So, for example, let x is 2 and the input output is 5 we want to find a neural network for relating this 2 and 5. So, we write the neural network and it has to be like this. We want that 5 should be in this particular manner. So, how to calculate this v 1 v 2 w 1 w 2 etcetera using the algorithm? So, initialize the weights to be v 1 v 2 is 0 0 and u 1 u 2 is also 0 initial weights.

Then the first iteration is denoted by 1 here in this super fix alpha is the learning rate is taken as 0.1 and the value is 2 times n. n is, when we substitute v 1 v 2 0 everything is 0 here in this y expression. So, we get 5 minus 0 actual value minus the neural network value at the 0 th iteration is 0 here. And, when we calculate that del E by del V 1 etcetera. It is coming out to be like this by substituting the 0 values. So, what we get here is the total value. The first iteration gives that v 1 is this v 2 is this and u 1 u 2 are 0 0.

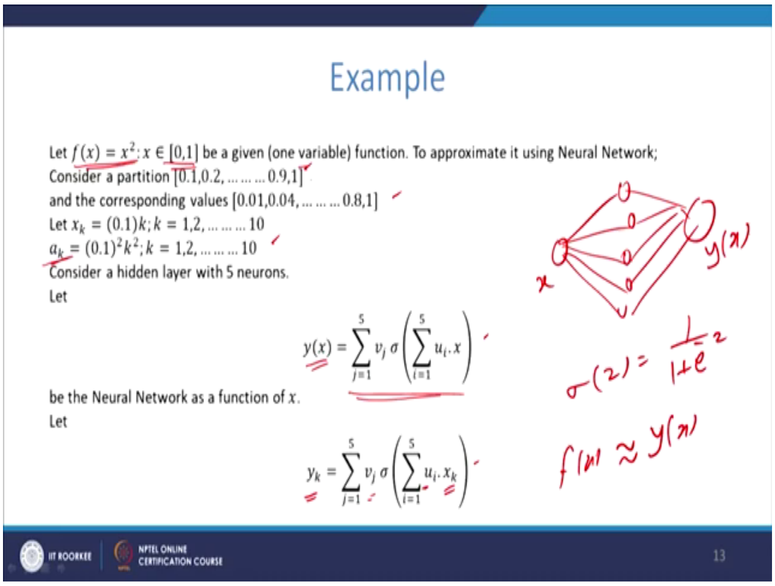
(Refer Slide Time: 15:33)



Now, the second iteration similarly can be calculated at second iteration the first iteration value minus alpha and 2 times the n value is this and the derivatives partial derivatives are given by this when we substitute the first iteration values. So, it gives the second iteration values to be like this. Now, if you substitute v 1 v 2 and u 1 u 2 in the neural network formula, we get the u close to 3.

So, we see that it has improved slightly the value of the neural network is coming towards the required value 5. So, proceeding like this for the third fourth iteration; we can easily see that the value of the neural network will converge towards 5 and it will be as closely as possible we can find the weights suitably.

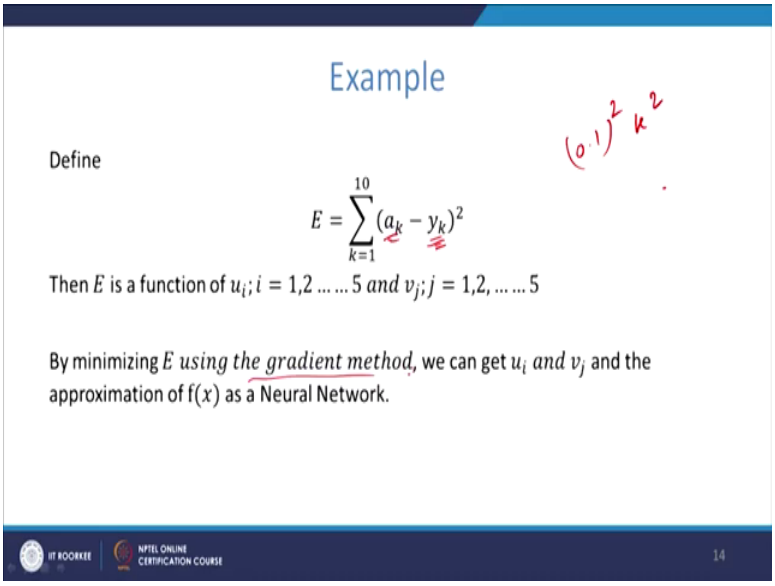
(Refer Slide Time: 16:36)



Now, if f of x is a given function. So, earlier we have seen how a constant output is approximated by a neural network. Now, if a function f of x is given in the interval 0 to 1 how to approximate it using a neural network. So, let us divide the interval by 0.1 0.2 etcetera up to 1 the given interval. And, the so the square of the x is x square is given by 0.01 and 0.04 etcetera 0.81. So, the square of this values are given here in this bracket. And so if we denote x k to be 0.1 into k that is given by this interval and the value alpha k; the required values are actually the square of this values these are the required values.

Now, if you define the neural network for any value of x to be in this particular manner. Here we are considering 1 input again we consider 5 hidden neurons in this case and then 1 output all of them together it is 1 output. So, this is x and the value is y of x and the calculation of the hidden neuron or given by this thing where sigma of z is taken to be 1 by 1 plus E power minus z as given in the previous one. Now, if we in the place of x, if we substitute the partition values 0.1 0.2 etcetera for x k. We can calculate for each value of the partition the corresponding neural network value is given by this one. So, if you suitably find this weights v i and u I, i equal to 1 2 3 up to 5.

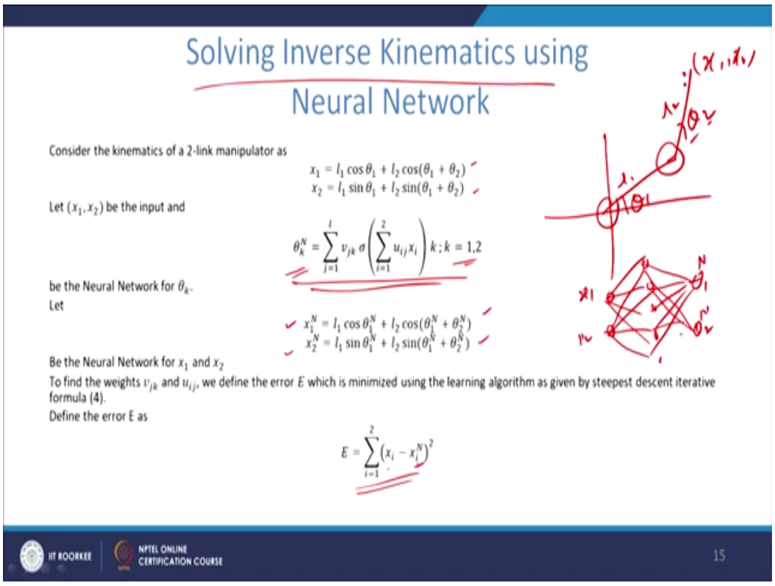
(Refer Slide Time: 18:58)



Then so, what is the requirement here; for each partition value the value a k is given by 0.1 square k square for k equal to 1 2 3 up to 10 and neural network values are given by the previous formula as given in this one. So, the difference between the required value alpha k and the neural network the square should be minimized at each and every partition point. So, error is summation k equal to 1 to 10 of the actual required value minus the neural network value whole square at each partition.

So, if you minimize this 1 using the gradient descent method has shown previously. We can easily obtain all the weight values. Now, if you substitute the converged value of the weights we get the function approximation that is, f of x is approximated by the function y of x by substituting the values which we obtain after many iterations of the artificial neural network algorithm. So, this can be; this is a function approximation using neural network. It is a simple example.

(Refer Slide Time: 20:32)



So, this procedure can be applied to solve inverse kinematics for any robot manipulator. So, for example, simple robot manipulator which we consider is the 2 link manipulator where theta 1 and theta 2 are the joint angles and x 1 and x 2 are the end effectors positions. So, x 1 is given by this and x 2 is given by this 1 where l 1 is the link length first link is l 1 and second link length is l 2.

So, we can easily solve the inverse kinematics. We can find theta 1 and theta 2 in terms of x 1 and x 2 very easily which we have seen in the previous lectures, but how to use the artificial neural network for finding the inverse kinematics solution is the following. So, let us consider the neural network value of theta k. Theta k, where k is 1 to 2, theta 1 n and theta 2 n is calculated by this neural network.

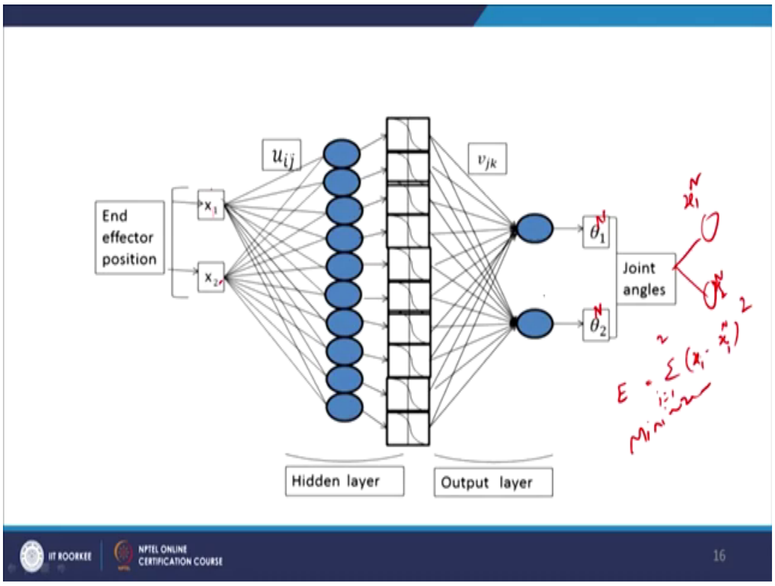
So, let us consider the input is here x 1 x 2. And, we consider hidden layers there are l hidden layers and we can connect the l hidden layers and with this and the output are given by this thing 2 outputs theta 1 and theta 2 are the outputs for this problem. So, this is coming out of the neural network therefore, we denote it by theta 1 n and theta 2 n. So, after finding suitable weights v i and w i when we substitute this theta the neural network values of theta should be very close to the actual inverse kinematic solutions of the problem.

So, how to find the weights properly? That is what we want to see. Now, when we initialize the weights arbitrarily, let us say all the weights are 0 in the beginning then we can substitute in the neural network and then obtain the theta values theta k n. Then substitute in this formula in the place of actual value of theta; we substitute neural network value of theta. And obtain the neural network value of the x x 1 and x 2 then we subtract the given value of x 1 x 2 minus the neural network value whole square.

So, this should be minimized E should be minimized suitably. So, that we get v i and u i values the weight values after conversions of the neural network algorithm. So, the error is given by this expression and using the gradient descent procedure algorithm. We can obtain the weights suitably and we get the after getting the final weights; we substitute in this particular formula again. So, whatever is coming out to be the neural network value that will be the actual value of the theta 1 and theta 2 the inverse kinematics solution of the problem.

So, this procedure can be adopted for any type of robot manipulator and for illustration purpose; we have taken the simplest form of this one.

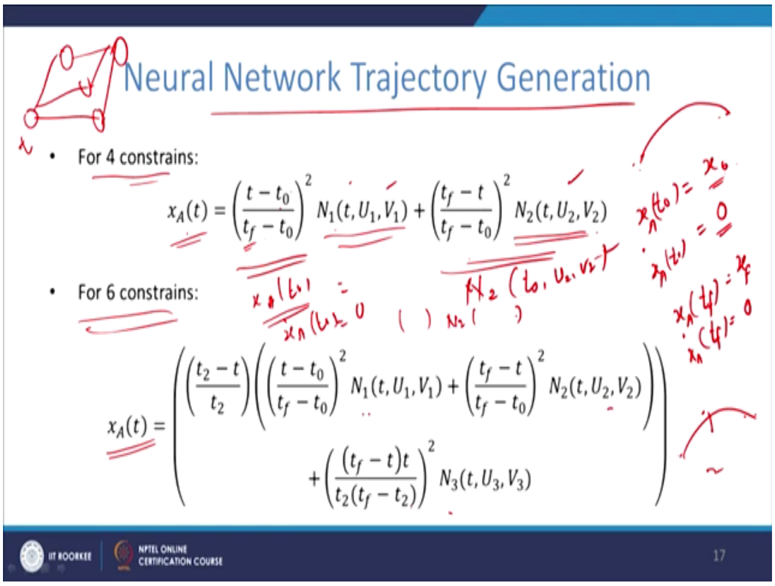
(Refer Slide Time: 24:46)



So, what we did here is; we have x 1 here it is written end effecter position and these weights are connected by u i j these are the hidden layer using the sigmoid function we calculate the theta 1 n and theta 2 n. So, these are the neural network output. So, once we get the joint angles using neural network we can again calculate from here x 1 n and x 2 n using the same formula then this will be taken back to this expression. The error is calculated the error is summation x i minus x i n whole square, i is equal to 1 to 2.

So, this is minimized. So, every time we get the inputs as x 1 and let us say x 2 and we get x 1 n and x 2 n and subtract and square it we minimize this expression to get the new weights. So, every iteration; we get a weight and put it here we keep on repeating finally, it will be converging to a required weight. So, that is a procedure.

(Refer Slide Time: 26:18)



Now, we can also generate the trajectories because we have seen the biped robot manipulator the ankle trajectory is there as hip trajectory. So, various trajectories are to be generated using polynomial. So, if there are let us say 4 constraints given initial condition initial velocity and final position and final velocity are given for a particular trajectory, then we can write a third degree polynomial and then we can find the coefficient of the polynomial using boundary conditions.

So, instead of that we can also write in this particular form at here t 0 denote the initial time t f denote the final time and. So, what it shows here is at t equal to t 0 the value is this bracket will vanish. So, now, if our condition is the ankles trajectory at t naught is given as some value let us say x naught and x A a t naught its derivative. The velocity is given to be 0, let us say and similarly x A t f is given to be x f and the velocity t f is some value for example 0.

So, if the initial position velocity final position final velocity are given for 4 conditions are given. Then we can generate a trajectory using a neural network. So, the neural network can be written as the as given in the previous slide. Here, the time is the input, t is the input and the value of the neural network is the output. And, let us say any number of neuron we can consider. So, the selection of the number of neuron in the hidden layer is left to the user here. So, normally for this type of problem we can consider 1 input we can take 3 or 4 hidden neurons and 1 output is given.

So, the value of the neural network after calculating the formula as given in the equation 3 is the value of the n 1 for this particular input t is the input. So, now, if you consider t equal to t 0; the first bracket is not there and here we have some constant value t f minus t. So, our aim here is only to train this N 2, the neural network 2 for this particular value that is x A at t naught equal to x naught.

So, when we substitute in the first formula x A at t naught that is nothing, but t f minus t naught whole square that is 1 and only the N 2 will come N 2 of t naught U 2 and V 2. The weight matrices U 2 and V 2 should be calculated for this particular value where the output of the neural network is given by x A of t naught which is equal to x naught that value should be taken. And, for this output we should find the weights U 2 and V 2 that is the meaning. Similarly we should differentiate this first equation x A dot t.

So, when we differentiate we will get some expression from the second term only. Because, when we substitute t naught after derivative also the first term vanishes only the second term will remain. So, in the second term; the value of the output is 0 when we differentiate x A dot t naught is 0, now after differentiating these 2 times this bracket multiplied by the neural network etcetera. And, then when we substitute t equal to t naught, we will get some constant value multiplied by the neural network.

So, our aim is to adjust the weights of the neural network for this two conditions because in the first condition also the neural network comes, in the second also we will get some bracket into the neural network. So, the neural network should be trained in such a way that it satisfies this two condition and the weights are calculated using the iterative method; because, the outputs are given and the U 2 and V 2 should be calculated. Similarly, for training this N 1 neural network; we substitute the final time t f.

When we substitute final time t f in the place of t we get the second term is 0 only the first term will survive. Similarly, when we differentiate it and then substitute t equal to t f the second term will become 0 only first term will remain. So, using these two conditions as output we should train the neural network 1 N 1 for finding the weights U 1 and V 1. Once we find U 1 V 1 U 2 V 2 using the iterative methods, we substitute we get the entire trajectory for all values of t. For training we will use only 2 time instances that is t equal to t 0 for training N 2 and t equal to t f for training N 1.

So, after getting all the weights; we substitute here that is useful for finding the trajectory for all values of t. So, similarly if we increase the number of conditions instead of initial and final we also have two conditions at the middle that is the height of that trajectory path as well as the velocity at this path the derivative. Then we can write the neural network in this particular form, 3 neural networks can be considered and by substituting the 3 time instances we can train according to the output values, the 3 neural networks and then substitute the weights here, we get the trajectory x A of t for the six constraints.

So, depending on the number of constraints, we can write the neural network expression and this can utilized as the trajectory for the biped robot walking, instead of the polynomial trajectory this can be also utilized. So, here so in this lecture we have seen how to utilize the neural networks for finding the inverse kinematics solutions for a robot manipulator as well as how to generate various trajectories for biped robot manipulators ok.

Thank you.