Intelligent Systems Control

Prof Laxmidhar Behera

Department of Electrical Engineering

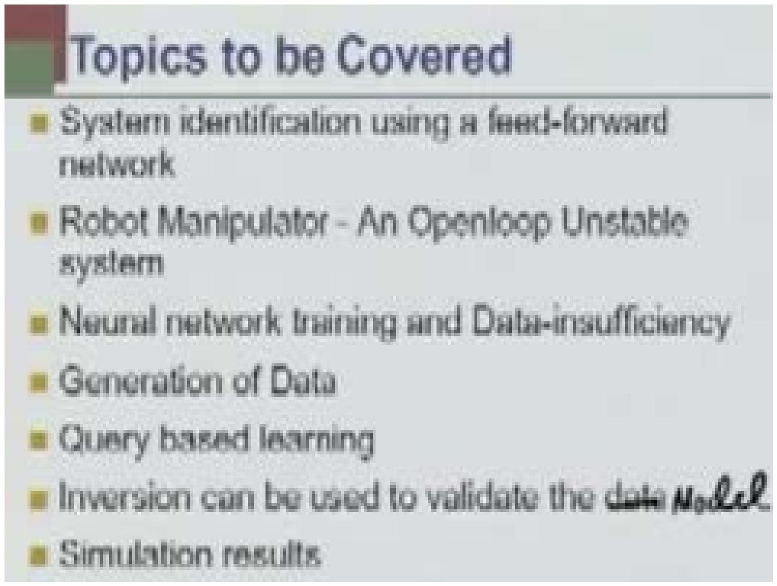
IIT, Kanpur

Module - 3 Lecture - 3
Neural Model of a Robot Manipulator

This is the third lecture in this third module on neural control in this course - intelligent control. Last class, we talked about concept of Network inversion and control and how can we compute what should be the control input, given the desired output, provided a neural network has been trained to learn a nonlinear dynamics of a given plant.

Today, we will be attacking the very foundation of this concept, because, network inversion depends on the model of the actual plant in the form of feed-forward neural network; whether it is a multilayered network or a radial basis function network. How good can a neural network be as a model of the plant? This is the question we will answer today.

(Refer Slide Time: 02:02)

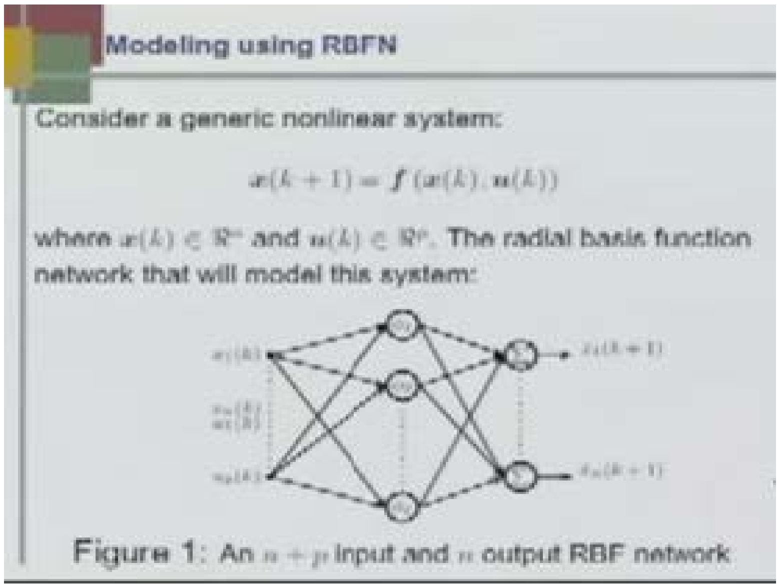


Topics to be Covered

- System identification using a feed-forward network
- Robot Manipulator - An Openloop Unstable system
- Neural network training and Data-insufficiency
- Generation of Data
- Query based learning
- Inversion can be used to validate the ~~data~~ Model
- Simulation results

The topics to be covered today are: system identification using a feed-forward network, robot manipulator; because, in the beginning of some lectures, we will focus on robot manipulators and many of the control application beyond robot manipulators, in this course. The robot manipulator is an open loop unstable system. The training of a network to learn a robot manipulator dynamically is very difficult. The neural network training and data insufficiency is a new concept that we will be talking today. Generation of data - when we have a problem, how do we generate data? We will be discussing a new scheme, a noble scheme of query based learning. Inversion can be used to validate the data, validate not the data and validate the model; finally, simulation results. We have already discussed some of these points in the last class, but it is worth always refreshing your memory on what we discussed last time.

(Refer Slide Time: 04:11)



Modeling using RBFN

Consider a generic nonlinear system:

$$x(k+1) = f(x(k), u(k))$$

where $x(k) \in \Re^n$ and $u(k) \in \Re^p$. The radial basis function network that will model this system:

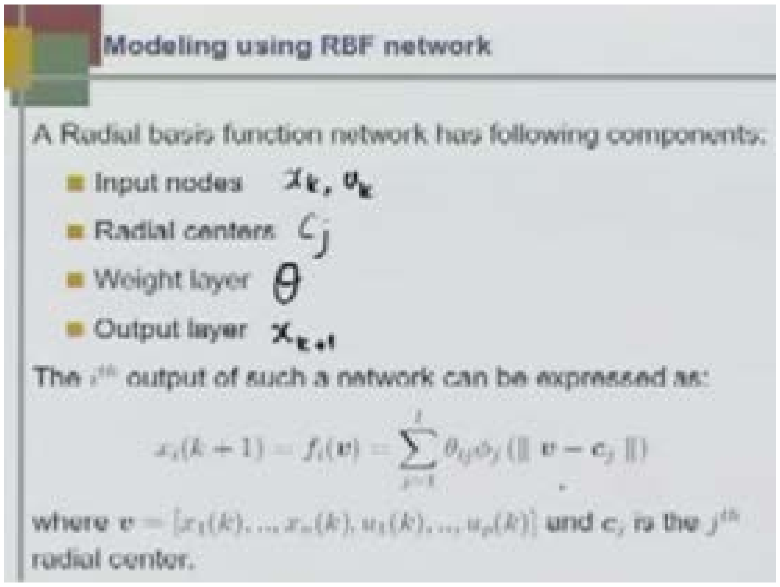
$x_1(k)$
$x_n(k)$
$u_1(k)$
$u_p(k)$
$\hat{x}_1(k+1)$
$\hat{x}_n(k+1)$

Figure 1: An $n+p$ input and n output RBF network

This is our non linear system in discrete time state space model. What you are seeing is that we have n state output, given the present n state and the present actuation. f is the nonlinearity, this is n by 1 vector. The function is modeled by this neural network, which is a radial basis function network. Although we have been discussing radial basis function network, whatever we are telling is also equally applicable to multilayered network - this is x_1 k to x_n k u_1 k to u t k ; so, these are the inputs to the neural network. This is the actual state of the plant and this is the actual input to the plant and this is the estimated output of the plant and the objective is that this estimated output of the radial basis function network should follow the actual plant response. This is what is desired. Then we can say this radial basis function network, is learnt (Refer Slide Time: 05:11).

As we have already said, the radial basis function network has the following components: input nodes; in this case, the input at the input nodes are: the current state vector x k and current control actuation vector u k , radial center which is represented by c $_j$ and weight vector, which is represented by theta and output layer; here input is x k and u k and this is x k plus 1 . The ith output of such a network can be expressed in this particular form, where this is my weight, this is the Gaussian activation. Usually, it can be any other basis function; thin plate function, inverse quadratic and so forth, this is known.

(Refer Slide Time: 06:15)



Modeling using RBF network

A Radial basis function network has following components:

- Input nodes x_k, u_k
- Radial centers c_j
- Weight layer θ
- Output layer x_{k+1}

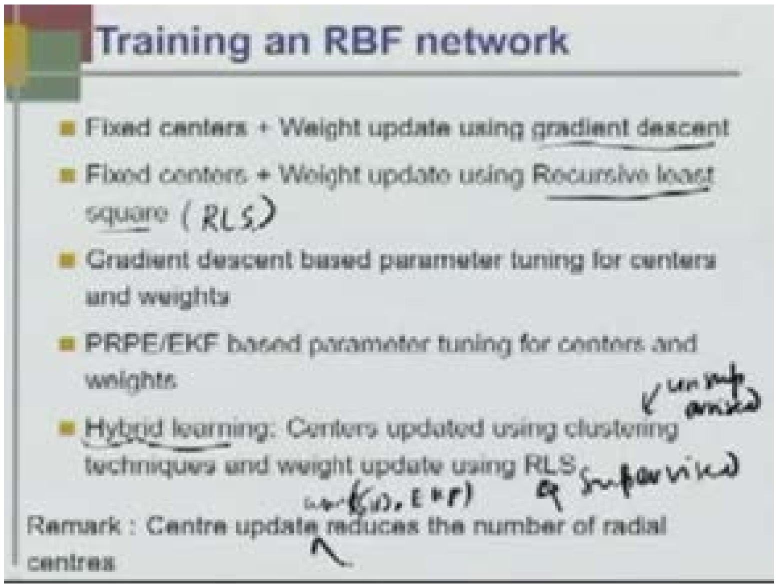
The i^{th} output of such a network can be expressed as:

$$x_i(k+1) = f_i(v) = \sum_{j=1}^{l} \theta_{ij}\phi_j(\| v - c_j \|)$$

where $v = [x_1(k), \ldots, x_n(k), u_1(k), \ldots, u_p(k)]$ and c_j is the j^{th} radial center.

V is actually this quantity, the vector of state as well as input and c $_j$ is the center, parameters, the reference vector associated with the center. This has a similar dimension as that of the input, so you can easily see v is x_1 k to x_n k u_n k to u $_p$ k ; c $_j$ is the j the radial center and the objective is that the radial basis function network output should match the actual plant output, given the present state of the actual plant and the present control actuation. Training of this network can be done in various manners. We can fix the centers uniformly distributed in the input space, and the weights can be updated using either gradient descent or using recursive least square, which is normally also very popularly known in the literature as RLS algorithm. The other approach is that we can use gradient descent to update both centers as well as the weight – theta in the output layer.

(Refer Slide Time: 08:11)



Training an RBF network

- Fixed centers + Weight update using gradient descent
- Fixed centers + Weight update using Recursive least square (RLS)
- Gradient descent based parameter tuning for centers and weights
- PRPE/EKF based parameter tuning for centers and weights
- Hybrid learning: Centers updated using clustering techniques and weight update using RLS

unsupervised

or supervised

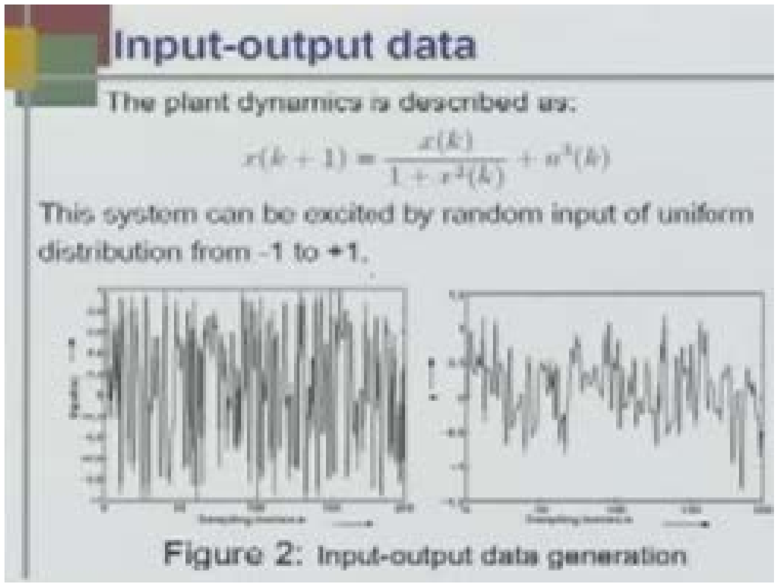
w.r.t (GD, EKF)

Remark : Centre update reduces the number of radial centres

Similarly, some advanced algorithm like Extended Kalman Filtering can be also used to update the parameters such as centers and weights. The other technique that also is very popular in radial basis function network is hybrid learning, where centers are decided by clustering method, which is unsupervised. This is unsupervised and weight update using recursive least square or gradient descent or E K F, whatever it is, this is supervised and hence this is hybrid learning. Normally, center update using either gradient descent or R L S gradient descent or E K F, reduces the number of radial centers. We will now go to the core of the system identification, how we really do the system identification. Given x k plus 1 as f x k and u k , what we would like to do is that we would first find proper architecture of a neural network that would fit to model this system. It is asked to learn the unknown nonlinear function f using a feed-forward network. It can be a either a multilayered network or radial basis function network.

What is the first step? The first step would be to generate data from the actual physical system, so we should have many examples already collected from the system and we will present those examples to this network that would model these dynamics. However, data generation becomes difficult if the physical system is open loop unstable. Before we discuss about open loop unstable system, we will consider open loop stable system - the simple one. This is a very simple nonlinear system, which is open loop stable, x k plus 1 is x k upon 1 plus x square k plus u cube k . You can easily see this is a nonlinear system, so we generate data.

(Refer Slide Time: 11:08)



Input-output data

The plant dynamics is described as:

$$x(k+1) = \frac{x(k)}{1+x^2(k)} + u^3(k)$$

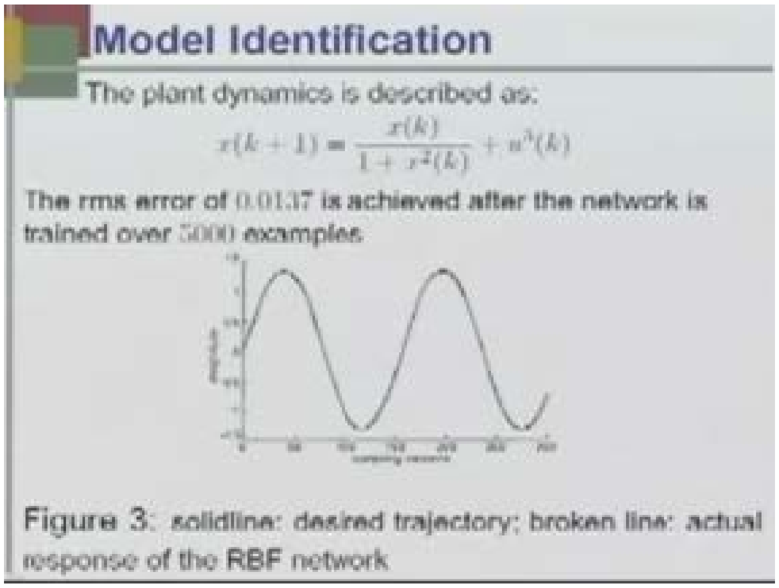
This system can be excited by random input of uniform distribution from -1 to +1.

Figure 2: Input-output data generation

What we have done here? Since this is an open loop unstable system, the system can be excited by random input of uniform distribution from minus 1 to plus 1.We can also take the distribution from minus 10 to plus 10. It is all up to you, how do you decide? In what dynamic range you want to operate the plant? That is up to you. Now, once I decide that my input will be from minus 1 to plus 1, I generate this random numbers. What you see- here, the first part, this is my input data and this is my corresponding output. This is my u k and this is x k plus 1 . Given u k this is my output k plus 1; you can easily see the output is bounded, almost between minus 1 point 5 to 1 point 5, where input is bounded between minus 1 to 1. This is a fairly stable system.

We will use these examples, these are 200 examples. We present these 200 examples to a radial basis function network which has this R B F N as radial basis function network. Obviously the input here will be u k as well as x k and your output is x k plus 1 . You have 2 inputs and 1 output system and what I said here is that you take a radial basis function network which has 2 inputs, u k and x k the output is x k plus 1 .

(Refer Slide Time: 13:13)



Model Identification

The plant dynamics is described as:

$$x(k+1) = \frac{x(k)}{1+x^2(k)} + u^3(k)$$

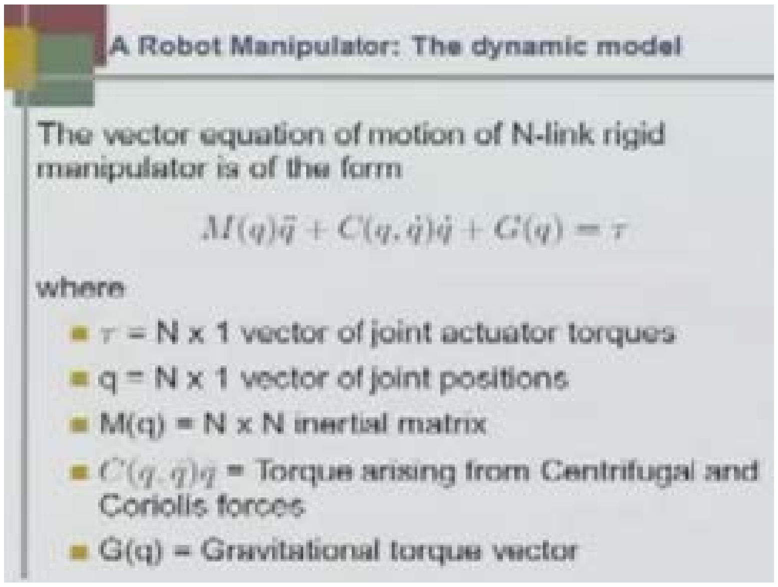
The rms error of 0.0137 is achieved after the network is trained over 5000 examples

Figure 3: solidline: desired trajectory; broken line: actual response of the RBF network

The training examples are already collected by simulation of this actual plant actual model, and then we trained this radial basis function network with 50 hidden units, for 5000 iterations. That is each training example has 200 data. If you look at 200, 25 times if I pass these examples through the network then, finally the convergence takes place and R M S error between. You can easily see here, this is our desired curve, the desired response, for which the network response is almost very similar; you can easily see a broken line, which is almost super imposed on the solid line. The solid line desired trajectory and the broken line which is almost superimposed on the solid line is actual response of R B F network. What you are seeing is that identifying a system dynamics for the simple systems are very fairly simple. They are not so difficult, only 5000 times if you train the network, it has a beautiful prediction capability. But today, in this lecture, normally what researchers have been doing is that given a plant and training examples, they train a network and after the network is trained, the network is placed or tested on a test data.

If the prediction is good, then we say ok network is learnt. But today we will utilize the notion of network inversion, to have a better definition or better validation mechanism for a system model. Soon we will learn today. Again as I said today's topic is neural model of a robot manipulator. We will be mostly dealing with robot manipulator during our control examples, so this is very important.

(Refer Slide Time: 15:02)



A Robot Manipulator: The dynamic model

The vector equation of motion of N-link rigid manipulator is of the form

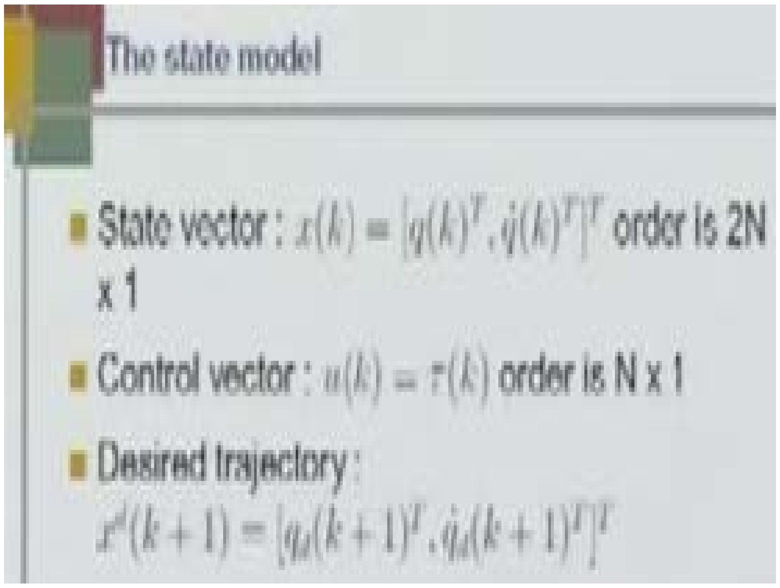
$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

where

- τ = N x 1 vector of joint actuator torques
- q = N x 1 vector of joint positions
- M(q) = N x N inertial matrix
- $C(q,\dot{q})\dot{q}$ = Torque arising from Centrifugal and Coriolis forces
- G(q) = Gravitational torque vector

You see that again N rigid link manipulator has this following nonlinear dynamics, where M q is the N by N inertial matrix q double dot is N by 1 acceleration vector, angular acceleration; C q q dot into q dot is torque raising from centrifugal and Coriolis forces the C is N by N matrix; q dot is N by one matrix vector G q is N by one vector, this is gravitational term. The total torque, the joint torque is applied to each joint, in the N link rigid manipulator, this is the torque vector. If you apply this torque at the joint actuation then the angular position and angular velocity of the robot link manipulator would follow this nonlinear dynamics. This is the model of a robot manipulator.

(Refer Slide Time: 17:00)



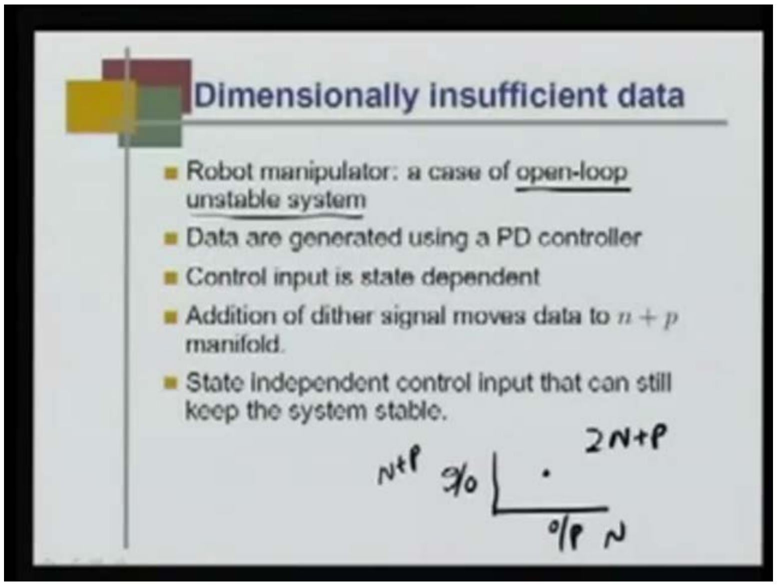
The state model

- State vector : $x(k) = [q(k)^T, \dot{q}(k)^T]^T$ order is 2N x 1
- Control vector : $u(k) = \tau(k)$ order is N x 1
- Desired trajectory : $x^d(k+1) = [q_d(k+1)^T, \dot{q}_d(k+1)^T]^T$

In the state space, we can define this robot manipulator which has 2 N states, because it has 6 position vector, 6 elements in the position vector, 6 elements in the velocity vector, N elements in the position vector and N elements in the velocity vector. 2 N total number of states control vector is N, and desired trajectory. It is required that robot manipulator links should follow the desired position vector q_d and desired velocity vector q_d dot. Earlier, we took a simple example and showed how easy it is to train a network that can learn such simple dynamics but when it comes to robot manipulators, we have problem. Why? The problem is that a robot manipulator is an open loop unstable system. Since it is an open loop unstable, how do we generate data? We have to use some minimal control mechanism that would guarantee some sense of stability. That is why data are generated using a PD controller.

We put a feedback loop and actuate a PD controller to generate the data. Control input thus a state dependent, because when I am taking the feedback from the output and computing the control action, the control action is function of state; hence it is a state dependent control action. What is the harm with that? I will explain that concept just in a moment.

(Refer Slide Time: 19:44)



Dimensionally insufficient data

- Robot manipulator: a case of open-loop unstable system
- Data are generated using a PD controller
- Control input is state dependent
- Addition of dither signal moves data to $n + p$ manifold
- State independent control input that can still keep the system stable.

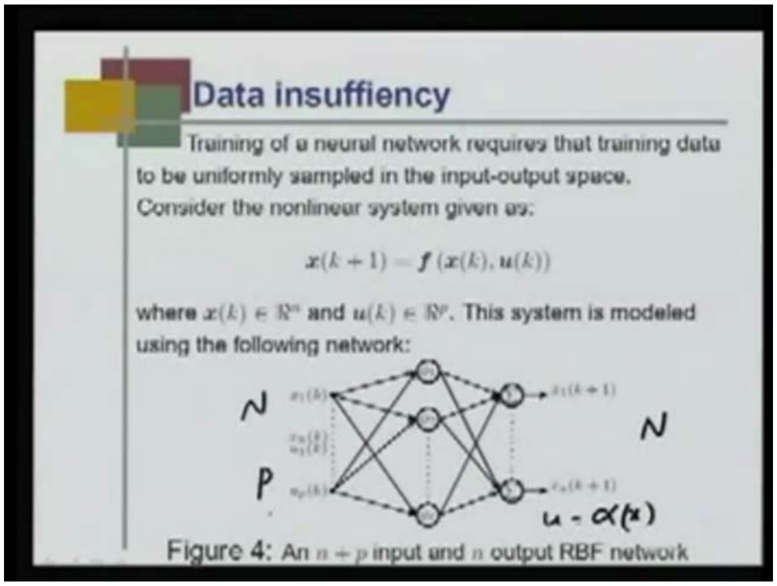
2N+P
N+P
i/o
o/p N

Now addition of dither signal moves data to n + p manifold. meaning is that what is happening is that I have this input dimension and this is my output dimension and this is my output space dimension of output space, and input dimension is N plus P and output dimension is N, my data. If I plot a data, this data is lying in 2 N plus P dimension. Why? Because finally this robot manipulator, which has 2N states. This is 2 N plus P states, this is 2 N, this is N, so because 2N states and N is the number of inputs, 2N is the output dimension. You can easily see this is 5N.

The input output dimension is 5N and in that I am looking at a point. A training example that I have generated is a point in 5N dimension in a space, which has a dimension 5N. Obviously if I am trying to train a network (Refer Slide Time 21:24), R B F N, which the output is 2N and input is 3N, when I am trying to train the network then the data must be sampled fairly uniformly from this total input-output space which has 5N dimensions. For the moment, I actuate the control action, which is a function of states. Obviously I have 2N states; my control u is a function of the states. What is happening is that, the data in which I am actually dealing is in 2N dimension, it is no more 4N dimension. By adding this dither signals, I can move this data to not n + p here, this is wrong, this is actually 3N dimension. So, still I am not able to reach the data points in 5N dimension. I will still further explain later. State independent control input that can still keep the system, so the requirement is that unless I generate control signals during data generation, which are state independent I will not be able to generate data that are fairly distributed over the complete 5 N dimensional manifold. This is called data insufficiency, and that means the bad news about any neural network is that you have to have a fair collection of data for a physical device model.

If I try to model any system, then the data generated from that system should be fairly distributed over the entire space, input output space.

(Refer Slide Time: 23:55)



Data insuffiency

Training of a neural network requires that training data to be uniformly sampled in the input-output space. Consider the nonlinear system given as:

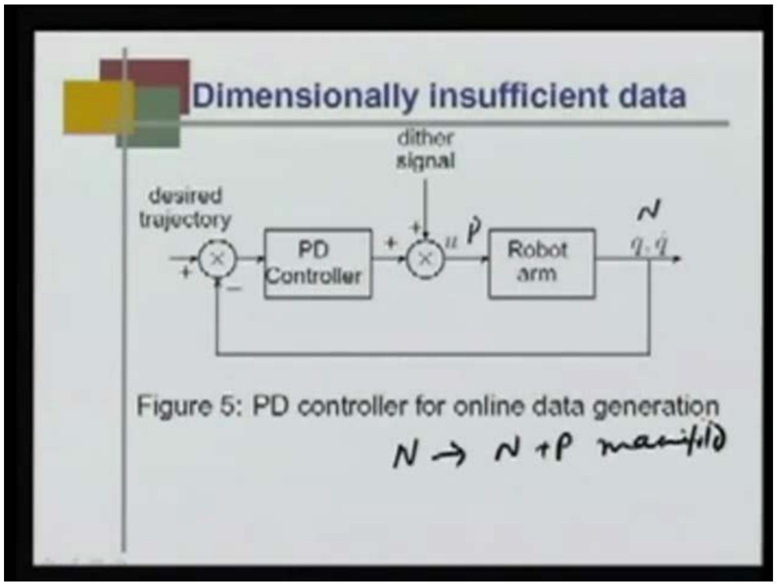
$$x(k+1) = f(x(k), u(k))$$

where $x(k) \in \mathbb{R}^n$ and $u(k) \in \mathbb{R}^p$. This system is modeled using the following network:

Figure 4: An $n+p$ input and n output RBF network

The training of a neural network requires that training data to be uniformly sampled in the input output space. Given this is our N dimension and this is my P dimension, when my input u is a function of x, obviously I lose because this x_1 k x_n k and x_1 k, and they are very fairly close. This is a function of alpha x, so obviously I am simply operating my data I am generating from the actual system plant using this controller, that is state dependent and I am only generating data that is in N dimension. What we can do to span that data to N plus P? What we can do is that, we can add some dither signal to P to the existing PD controller. By adding that, what we are trying to do?

(Refer Slide Time: 25:25)



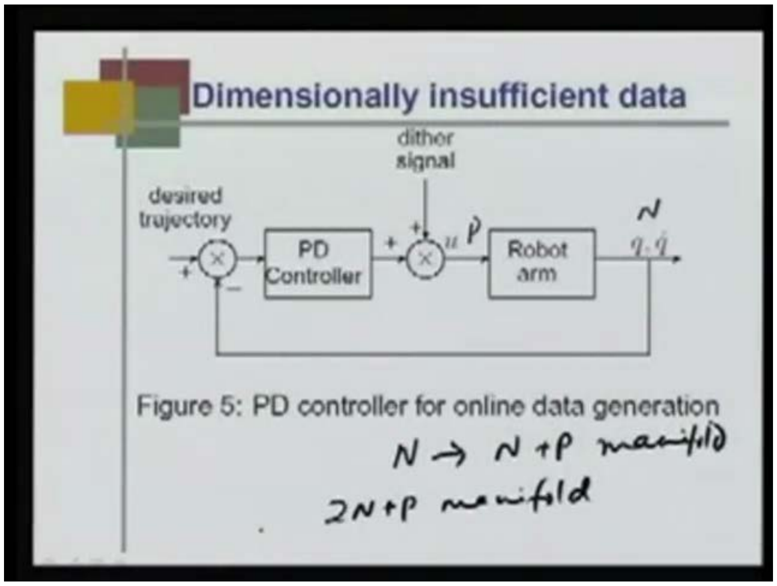
Dimensionally insufficient data

Figure 5: PD controller for online data generation

$N \rightarrow N + P$ manifold

By adding this dither signal here, this is my N states If I say this is my N states, and this is my P inputs, if I do not add this dither signal, I am operating in a manifold of N dimension. But by adding dither signal I am trying to expand this N dimensional manifold to N plus P manifold. That is the bad part of generating data for any open loop unstable system because we have to use some kind of controller, otherwise system will become unstable. When system becomes unstable, we have a problem. What you are seeing is that, this is a PD controller for online data generation, and I explained to you what is meaning of dimensionally insufficient data.

(Refer Slide Time: 26:44)



Dimensionally insufficient data

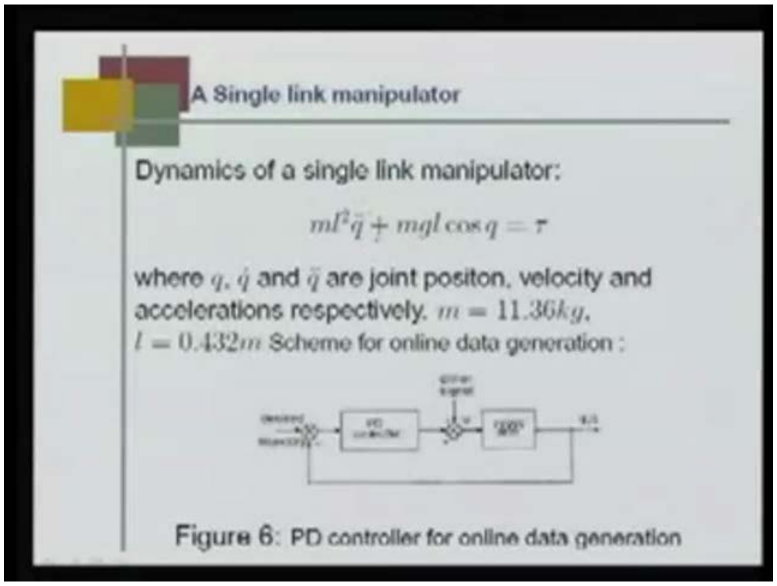
Figure 5: PD controller for online data generation

$N \rightarrow N+P$ manifold

$2N+P$ manifold

Dimensional insufficient data means, when we train a neural network of a system having N states and P inputs, then we are operating in 2N plus P manifold. We must have training examples that should be generated, which should be fairly distributed over this 2N plus P manifold. Instead, the mechanism that we have for generation in usual practice, that can maximally span to N plus P manifold. It is never as close to N plus P from N to N plus P it can just expand.

(Refer Slide Time: 27:18)



A Single link manipulator

Dynamics of a single link manipulator:

$$ml^2\ddot{q} + mgl\cos q = \tau$$

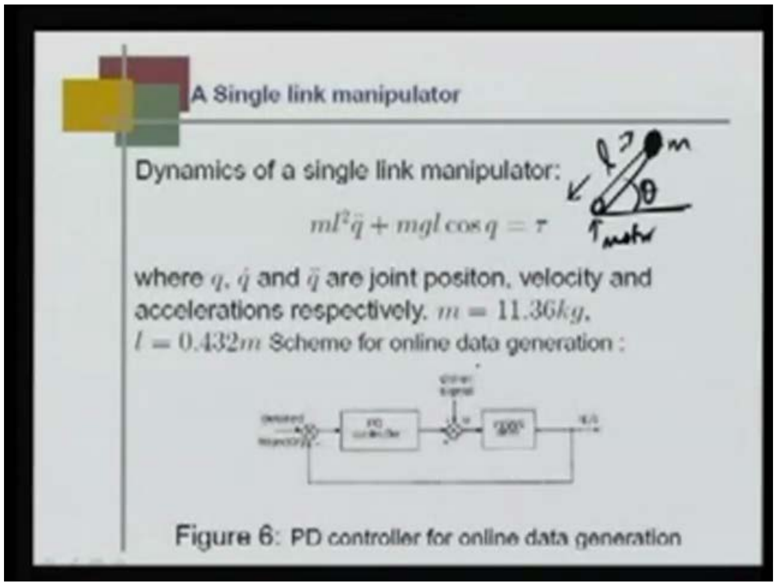
where q, $\dot{q}$ and $\ddot{q}$ are joint positon, velocity and accelerations respectively. $m = 11.36kg$, $l = 0.432m$ Scheme for online data generation :

desired trajectory
PD controller
dither signal
robot arm

Figure 6: PD controller for online data generation

We will further understand this concept of dimensionally insufficient data, by taking the example of a single link manipulator. The dynamics of a single link manipulator is given as m l square q double dot plus m g l cos q is torque, where this is my single link manipulator, and this is my horizontal axis and this is my theta. This is the motor here, that and this is the mass, which is point mass situated at the end point of the manipulator. Assume that m is (Refer Slide Time: 11:36) this l is 0 point 432 meter. Given this, I generate data using the same scheme that we just discussed - that we have a PD controller and we add some other signals.

(Refer Slide Time: 28:47)



A Single link manipulator

Dynamics of a single link manipulator:

$$ml^2\ddot{q} + mgl\cos q = \tau$$

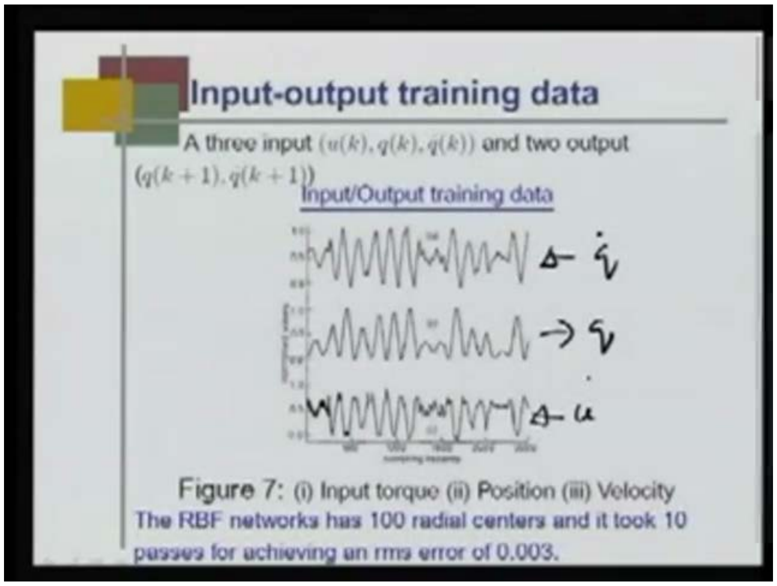
where q, $\dot{q}$ and $\ddot{q}$ are joint positon, velocity and accelerations respectively. $m = 11.36kg$, $l = 0.432m$ Scheme for online data generation :

l
m
θ
motor
desired trajectory
PD controller
dither signal
u
q, q̇

Figure 6: PD controller for online data generation

What is dither signal? Dither signal is actually signal that the noise, white noise, state signal, ramp, parabolic type of signal, whatever the signals that you can add to this PD controller output.

(Refer Slide Time: 29:03)



Input-output training data

A three input $(u(k), q(k), \dot{q}(k))$ and two output $(q(k+1), \dot{q}(k+1))$

Input/Output training data
$\dot{q}$
q
u

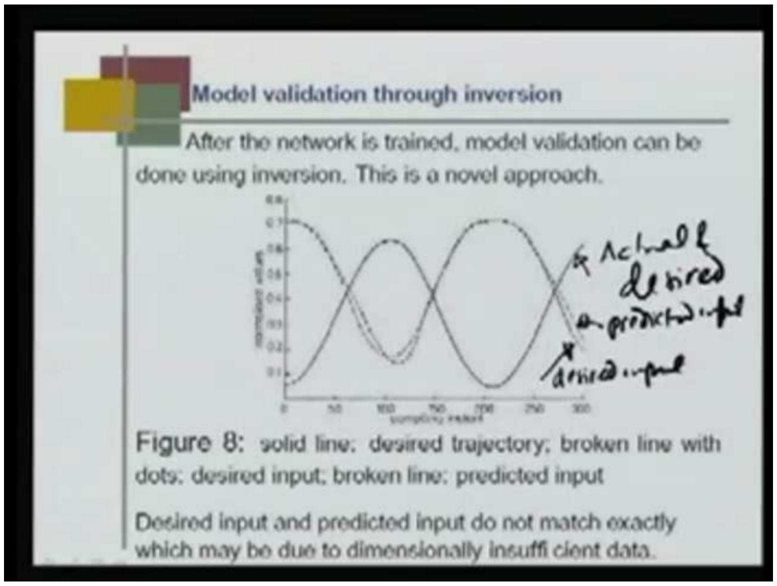
Figure 7: (i) Input torque (ii) Position (iii) Velocity

The RBF networks has 100 radial centers and it took 10 passes for achieving an rms error of 0.003.

You see that here a typical training data set or training examples that were generated for a robot manipulator, single link robot manipulator, this is our input torque or u.

This is the position vector q and this is q dot, the velocity vector because the single link manipulator, we have only single angular position, single angular velocity and single joint torque. The R B F network that has 100 radial centers, which as 3 inputs u k q k and q dot k and 2 output, which is q k plus 1 and q dot k plus 1 ; this is the structure of R B F network, 100 radial centers. It took 10 passes, actually this number of data points, running examples were actually 3000 training examples and these 3000 training examples, over 10 passes means 30000 iterations. That many training we got an R M S, fairly good R M S error called 0 point 003, between the desired target point or desired trajectory and actual response. After training is over usual practice is that we give a test data and then we see whether the desired trajectory is matching with the actual response. But in this case we do both.

(Refer Slide Time: 31:02)


Model validation through inversion

After the network is trained, model validation can be done using inversion. This is a novel approach.

Actual & desired
predicted input
desired input

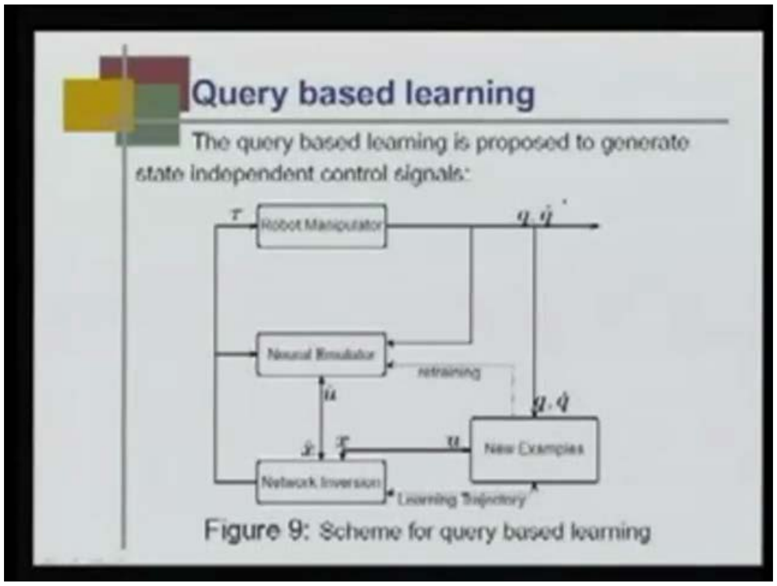
Figure 8: solid line: desired trajectory; broken line with dots: desired input; broken line: predicted input

Desired input and predicted input do not match exactly which may be due to dimensionally insuffi cient data.

You can easily see that the desired trajectory and actual trajectory are - this particular thing is both actual and desired. One may be very happy to say, “oh my network has 100 percent learnt the system dynamics”, but this is not true. You can see that there are two other curves in this figure. So, what you are seeing is that this particular curve is actual and desired trajectory, and here, this particular curve is the broken line with dots, is the desired input. This is the desired input and this is the predicted input. For a control system it is very fair because, what we do is, given a control plan and a control model, we give control actuation, given a specific control actuation, then we look at the desired trajectory. And same control actuation we give to the radial basis function network and see what is the radial basis function network output. If they match then we say it is a good model, which already established in the first case. If I now ask this question to this radial basis function network - please tell me given this desired trajectory, what is the control input? Which is this predicted input? You see that predicted input and desired input although they match there is some dissimilarity you see. They exactly do not match the predicted and actual they do not exactly match.

It is a fair degree of matching, but still they are not a good match. This is a single link manipulator, very simple system, but still there is some disparity. Due to this disparity, we can say that this disparity comes because of the insufficient data. To generate dimensionally sufficient data, for any open loop unstable system, is very difficult. For this we would now talk about query based learning. What we saw is that although actual and desired trajectory, they match fairly accurately. In which respect they do not match is that desired given actual trajectory to the network; what is the input? Predict the desired input. There is difference between desired input and predicted input - this we say model validation through inversion. This is the better way to validate our model, because for dynamical system, both questions are important. Given input what is the output? Given an output what is the input? This is very important. For that - we are now proposing how we can generate dimensionally sufficient data. As you see in the beginning, we told that to generate dimensionally sufficient data means we have to have control actuation which is independent of states. You saw that it is very difficult to control actuation which is independent of control states.

(Refer Slide Time: 35:06)



Query based learning

The query based learning is proposed to generate state independent control signals:

Figure 9: Scheme for query based learning

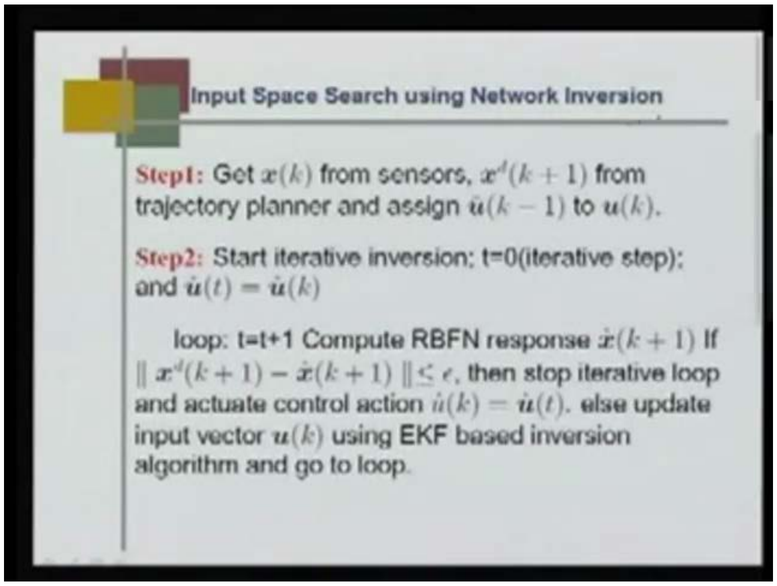
Here is a method that is proposed for generating dimensionally sufficient data, so what you are seeing is that, this I say query based learning. What is query based learning? Query based learning is proposed to generate state independent control signals. It is very important this concept, to generate state independent control signals. This is the robot manipulator; we actuate a torque.

In general we actuate this torque using PD controller, but now instead of PD controller, what we are trying to do is, that robot manipulator is already trained from initial training data. What we followed earlier is called neural emulator, or neural model of the manipulator. We have already had some form of neural emulator from the previous training examples, and those training examples were generated using PD controller. But once we have a preliminary version of a neural emulator, we put the emulator and put a network inversion algorithm here. What this network inversion algorithm does is that given a setup, some learning trajectory, while we generated some data, we generate a data over some trajectory. We will create some new trajectories and along this new trajectory we will invert the network, and ask this question, what is u? What is the control input for a desired trajectory? If my answer is in negative that means, if I apply that torque I do not get that specific trajectory, then I store that as a new training example. I hope you are able to follow what I am saying.

I present a new learning trajectory to the robot manipulator now. Along that new trajectory I do the network inversion. To this network inversion, given a desired trajectory, defined by learning trajectory, what should be my u? I give that u to the robot manipulator, see what my state is. If my state is not the desired state, the learning trajectory, then I put those data in the new example set and I go on and whenever my input exactly predicts what my target for a current state is, I reject those examples. In this way I collect in a basket or in a database more and more new examples, and I utilize these new examples for retraining the neural emulator. A parallel processing is going on where I see in which zone the network is not trained, this is all completely online. This entire scheme is an online scheme. In online we can do this, we can experiment, and we can experimentally validate our scheme. To summarize, this query based learning is very important for you to understand, the query based learning is proposed to generate state independent control signals.

You see that in this particular charter or plan, we do not have a PD controller. Instead what we have introduced using the previous training example, the trained neural network, which we say the preliminary version of the neural emulator, this is placed in parallel with the plant and we place the network inversion algorithm, we provide new learning trajectories and along each learning trajectory, we find those regions in the input output space where the neural emulator does not have proper training. That is how do we do it? Given a desired trajectory, we invert the neural network, neural emulator and we make a decision as to what should be the control action. The same control action is actuated with the actual plant and then we find out what is the actual trajectory. We compare this actual trajectory with the desired trajectory and if they do not match then we discard and take that as a new example. If they match then we discard. This way we create more and more new examples in those zones where the neural emulator is not properly trained and we retrain the neural emulator. This is called retraining.

(Refer Slide Time: 40:28)



Input Space Search using Network Inversion

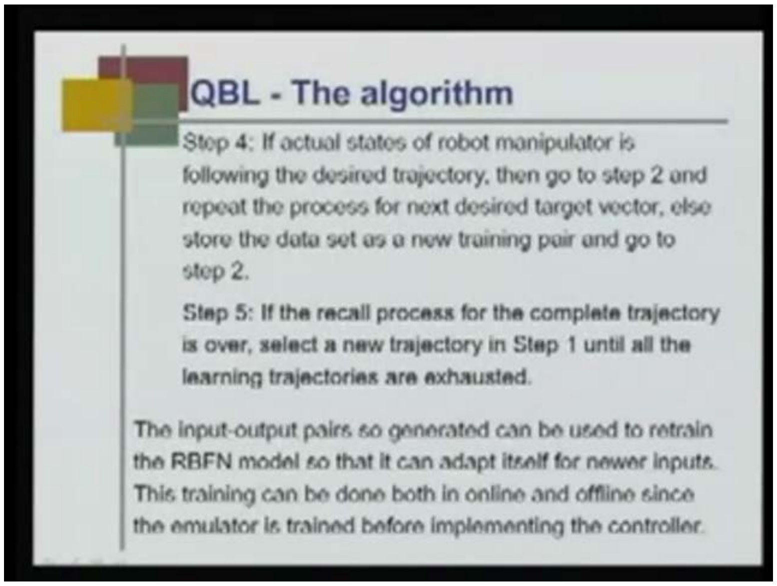
Step1: Get $x(k)$ from sensors, $x^d(k+1)$ from trajectory planner and assign $\hat{u}(k-1)$ to $u(k)$.

Step2: Start iterative inversion: t=0(iterative step); and $\hat{u}(t) = \hat{u}(k)$

loop: t=t+1 Compute RBFN response $\hat{x}(k+1)$ If $\| x^d(k+1) - \hat{x}(k+1) \| \leq \epsilon$, then stop iterative loop and actuate control action $\hat{u}(k) = \hat{u}(t)$. else update input vector $u(k)$ using EKF based inversion algorithm and go to loop.

This is the input space search using network inversion, which we have already talked earlier and this refining - what is network inversion? Get x k from the sensors x t k plus 1 from the trajectory planner, assign the input previous in-control actuation to represent control actuation and then start iterative inversion, t equal to t plus 1, so computer R B F N responds x k plus 1 , given x k and the initial value of the control actuation. If the network prediction is same as the desired target, then we stop the iteration. Otherwise we update the input actuation using either E K F or gradient search or Lyapunov function. But in the last class, we saw that E K F better that is why here it is written only E K F. This is query based learning. The complete algorithm is like this: select a new training trajectory from a set of finite number of learning trajectories, spanning over the robot work space, for a desired target vector selected sequentially as per the learning trajectory, compute control input by inversion of radial basis function model to get the desired output taking present state into account, actuate the control action to robot manipulator and observe joint positions and velocities.

(Refer Slide Time: 43:00)



QBL - The algorithm

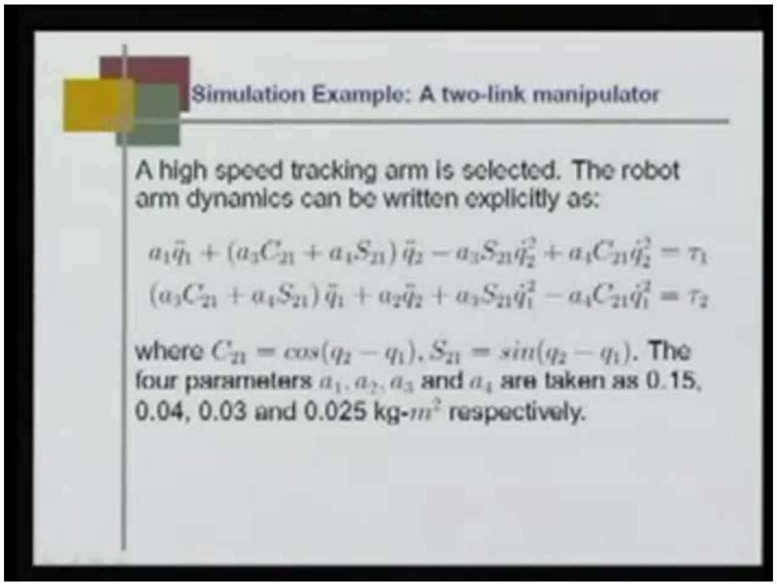
Step 4: If actual states of robot manipulator is following the desired trajectory, then go to step 2 and repeat the process for next desired target vector, else store the data set as a new training pair and go to step 2.

Step 5: If the recall process for the complete trajectory is over, select a new trajectory in Step 1 until all the learning trajectories are exhausted.

The input-output pairs so generated can be used to retrain the RBFN model so that it can adapt itself for newer inputs. This training can be done both in online and offline since the emulator is trained before implementing the controller.

If actual states of the robot manipulator is following the desired trajectory, then go to step 2 and repeat the process for next desired target vector; else, store the data set as a new training pair and go to step 2. Step 5 if the recall process for the complete trajectory is over; select a new trajectory in step 1 until all the learning trajectories are exhausted. This is where the query based learning algorithm is implemented the input output pairs so generated can be used to retrain the radial basis function network model so that it can adapt itself for newer inputs. This training can be done both in online and offline since the emulator is trained before implementing the controller. Now we will talk about a new way, a noble way of validating a neural model of a robot manipulator, and that concept is through network inversion.

(Refer Slide Time: 44:00)



Simulation Example: A two-link manipulator

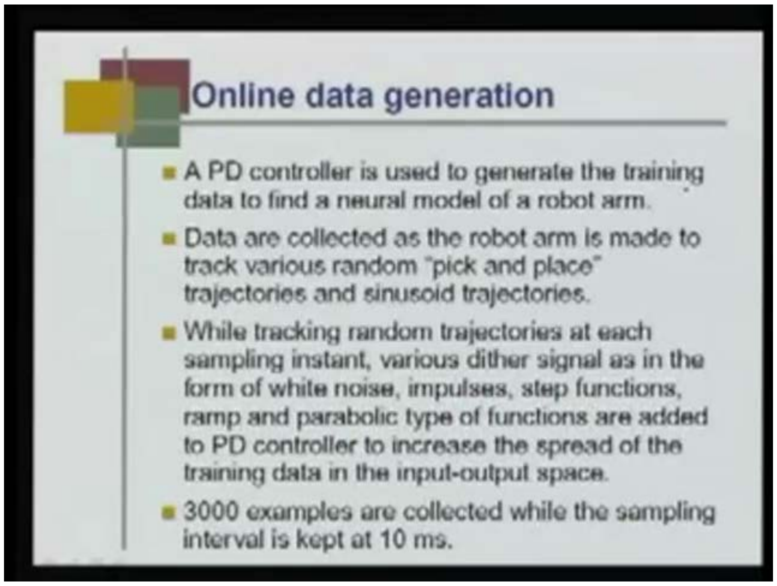
A high speed tracking arm is selected. The robot arm dynamics can be written explicitly as:

$$a_1\ddot{q}_1 + (a_3C_{21} + a_4S_{21})\ddot{q}_2 - a_3S_{21}\dot{q}_2^2 + a_4C_{21}\dot{q}_2^2 = \tau_1$$
$$(a_3C_{21} + a_4S_{21})\ddot{q}_1 + a_2\ddot{q}_2 + a_3S_{21}\dot{q}_1^2 - a_4C_{21}\dot{q}_1^2 = \tau_2$$

where $C_{21} = cos(q_2 - q_1)$, $S_{21} = sin(q_2 - q_1)$. The four parameters a_1, a_2, a_3 and a_4 are taken as 0.15, 0.04, 0.03 and 0.025 kg-m^2 respectively.

We will see that; we will be first learning the model. We will be training a network radial basis function network that will learn the dynamics of a 2 link manipulator. The same, earlier we have also considered this 2 link manipulator. A high speed tracking arm is selected. This robot arm can move very fast, the robot arm dynamics is written explicitly as follows: you can easily see here, where $C_{2\ 1}$ is cos q_2 minus q_1, and $S_{2\ 1}$ is sine q_2 minus q_1, so here q_1 and q dot are q_1 dot and $q_{2,}$ q_2 dot double dot and q_1 double dot there acceleration, parameters $a_{1,}$ $a_{2,}$ $a_{3,}$ a_4 are 0.00, 0.15, 0.04, 0.03 and 0.25 k g meter square. First, we have to generate data to find an equivalent radial basis function network.

(Refer Slide Time: 45:37)



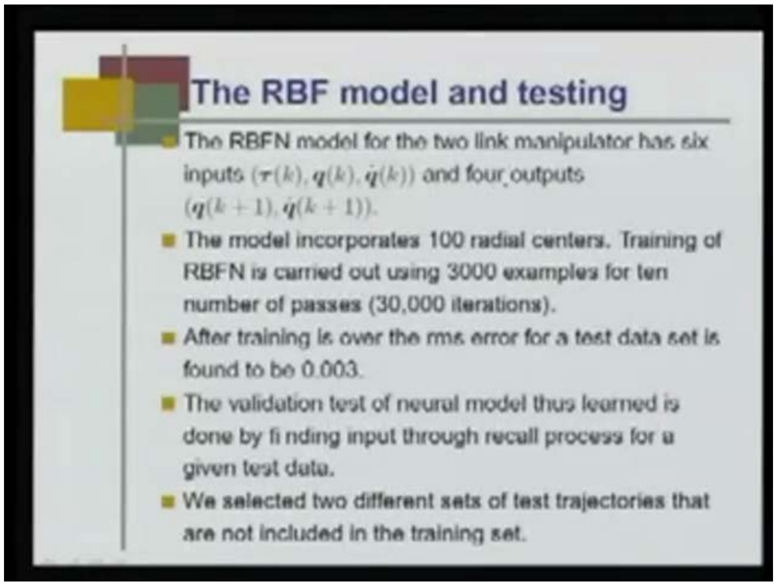
Online data generation

- A PD controller is used to generate the training data to find a neural model of a robot arm.
- Data are collected as the robot arm is made to track various random "pick and place" trajectories and sinusoid trajectories.
- While tracking random trajectories at each sampling instant, various dither signal as in the form of white noise, impulses, step functions, ramp and parabolic type of functions are added to PD controller to increase the spread of the training data in the input-output space.
- 3000 examples are collected while the sampling interval is kept at 10 ms.

A PD controller is used to generate the training data to find a neural model of the robot arm. Data are collected as the robot arm is made to track various random pick and place trajectories and sinusoid trajectories.

What we try to do through PD controller to take the robot manipulator along either pick and place trajectory that is from one set point to another set point or a sinusoid trajectory. While tracking random trajectories, at each sampling instant, various dither signals in the form of white noise, impulses, step functions, ramp and parabolic type of functions are added to PD controller to increase the spread of the training data in the input output space. 3000 examples are collected while the sampling interval is kept at 10 milliseconds.

(Refer Slide Time: 46:56)

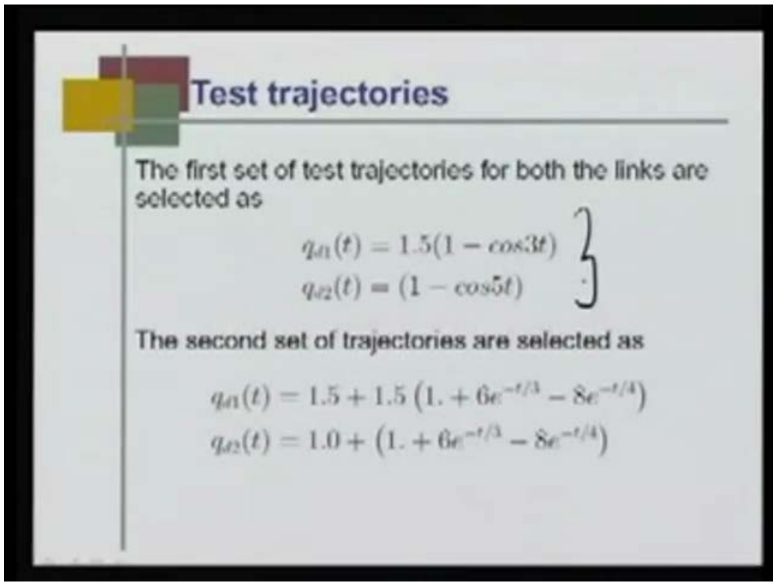


The RBF model and testing

- The RBFN model for the two link manipulator has six inputs $(\boldsymbol{\tau}(k), \boldsymbol{q}(k), \dot{\boldsymbol{q}}(k))$ and four outputs $(\boldsymbol{q}(k+1), \dot{\boldsymbol{q}}(k+1))$.
- The model incorporates 100 radial centers. Training of RBFN is carried out using 3000 examples for ten number of passes (30,000 iterations).
- After training is over the rms error for a test data set is found to be 0.003.
- The validation test of neural model thus learned is done by fi nding input through recall process for a given test data.
- We selected two different sets of test trajectories that are not included in the training set.

The R B F model and testing, the radial basis function model for the 2 links has 6 inputs, because you have 2 joint torques, 2 angular positions and 2 angular velocities and 4 output, 2 angular positions and 2 angular velocities. The model incorporates 100 radial centers. Training of radial basis function network is carried out using 3000 examples for 10 number of passes and that means 30000 iterations after training is over the R M S error for a test data set is found to be 0.003. The validation of test of neural model thus learnt is done by finding input through recall process for a given test data. This is the noble way of testing whether your model is accurate or not. This is called model validation through inversion; we selected two different sets of test trajectories that are not included in the training set.

(Refer Slide Time: 47:48)



Test trajectories

The first set of test trajectories for both the links are selected as

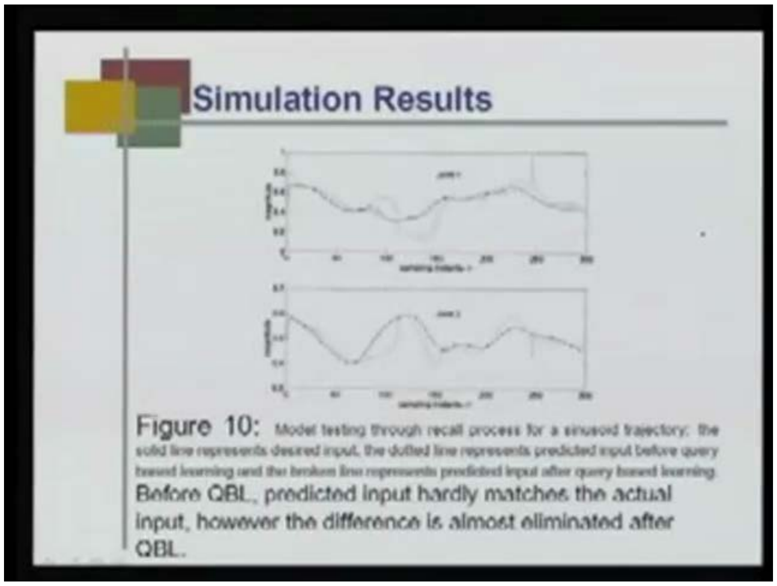
$$\left.\begin{aligned} q_{d1}(t) &= 1.5(1 - cos3t) \\ q_{d2}(t) &= (1 - cos5t) \end{aligned}\right\}$$

The second set of trajectories are selected as

$$q_{d1}(t) = 1.5 + 1.5\left(1. + 6e^{-t/3} - 8e^{-t/4}\right)$$
$$q_{d2}(t) = 1.0 + \left(1. + 6e^{-t/3} - 8e^{-t/4}\right)$$

These are the two. This is the first set of training trajectory; this is the second set of training trajectory. This is test trajectory 1 and test trajectory 2. You can see the first one is sinusoid and second one is exponential. This is the first part; what you are seeing here is that the desired trajectory is sinusoid. If I were to simply compare - given the input, what is the desired trajectory? The sinusoid trajectories; then from the test data - always the radial basis function network predicts almost very accurately the R M S error is 0 point 003. So, the desired trajectory and actual trajectory match almost very closely.

(Refer Slide Time: 49:02)



Simulation Results

Figure 10: Model testing through recall process for a sinusoid trajectory: the solid line represents desired input, the dotted line represents predicted input before query based learning and the broken line represents predicted input after query based learning.

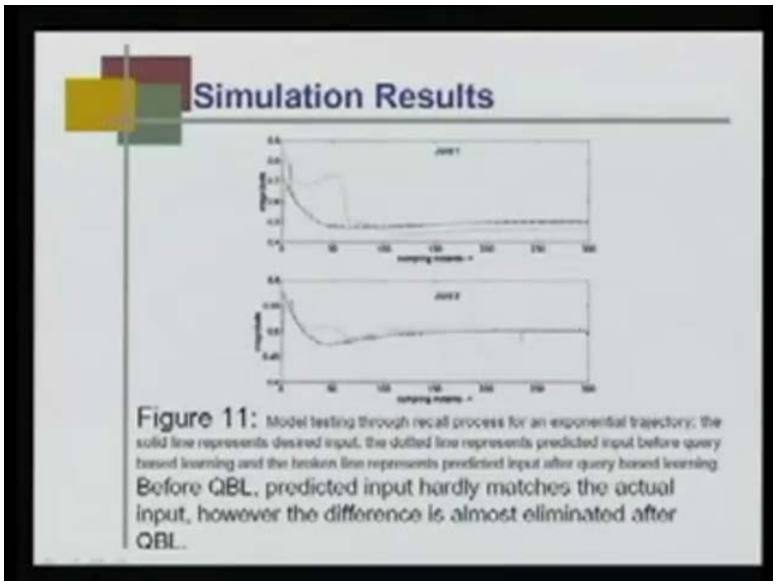
Before QBL, predicted input hardly matches the actual input, however the difference is almost eliminated after QBL.

If I really do inversion then what will happen here? You see that what we did here are two things - first we took 3000 data set and we tried. After taking 3000 data set, we saw that actual trajectory, the response of the radial basis function network and desired trajectory and they match. But if I invert, if I ask the question, given the desired trajectory, what is the control input? You see the solid one; this is the actual control input that is given to the robot manipulator and this dotted one which you are seeing here, this pen tip is moving along this line, this particular one is the predicted input. You see the predicted input is very far away from the desired input. Because of the inversion, I have two different data points. I can look how close it is at the input and how close it is at the output. It is both way validations; so, normally what happens is when we check the validity of a neural model we always see at the output, we do not see at the input. But this validation what you are seeing, that I am showing the validation at the input. You see at the input, the validation is not there in the beginning. What I did after 3000 iterations is, I took that neural emulator and used the query based learning and I further trained the network. After the query based learning you see the input predicted.

The input predicted is now very close to the actual one, very close to the desired one. That is now because of query based learning now. Not only the network is able to predict the desired trajectory for a given desired input, but also for a desired trajectory it is able to predict what the desired input is. Both way predictions is happening. That is much better and accurate model of system dynamics than what it used to be without query based learning. Without further explaining you can see, this is for joint 1 which I showed you. Now for the joint 2 also, before query based learning, the predicted input is very far away from the actual or desired input and after query based learning, you can easily see that the predicted and desired are almost very close.

To summarize this, model testing through recall process for a sinusoid trajectory, the solid line represents desired input. The dotted line represents predicted input before query based learning, and the broken line represents predicted input after query based learning. Before query based learning, predicted input hardly matches the actual input. However, the difference is almost eliminated after query based learning. This is the first test trajectory.

(Refer Slide Time: 53:14)



Simulation Results

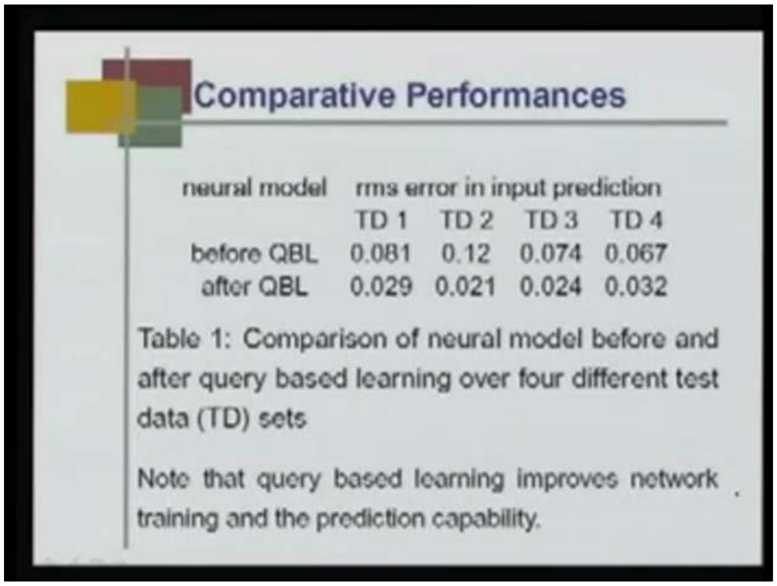
Figure 11: Model testing through recall process for an exponential trajectory; the solid line represents desired input, the dotted line represents predicted input before query based learning and the broken line represents predicted input after query based learning

Before QBL, predicted input hardly matches the actual input, however the difference is almost eliminated after QBL.

This is the second test trajectory. In the second test trajectory, the trajectory was exponential as I said earlier and you see that solid line is the desired control input. If I invert before query based learning, then you see that the predicted control signal is very far away from the actual or desired control input. After query based learning, you see that the desired and the predicted one almost match. This is for joint 1, this is for joint 2, again a similar thing, we can again summarize.

Model testing through recall process for an exponential trajectory the solid line represents desired input, the dotted line represents predicted input before query based learning, and the broken line represents predicted input after query based learning. Before query based learning predicted input hardly matches the actual input. However the difference is almost eliminated after query based learning.

(Refer Slide Time: 54:18)



Comparative Performances

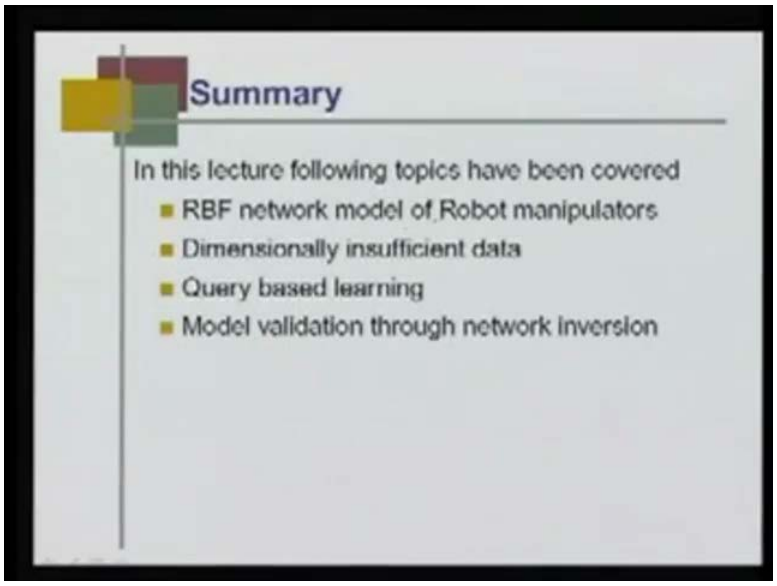
neural model	rms error in input prediction			
	TD 1	TD 2	TD 3	TD 4
before QBL	0.081	0.12	0.074	0.067
after QBL	0.029	0.021	0.024	0.032

Table 1: Comparison of neural model before and after query based learning over four different test data (TD) sets

Note that query based learning improves network training and the prediction capability.

The comparative performance, if we do for 4 test data, then we have taken 4 test data here, then you see that the R M S error before query based learning, used to be very high and after query based learning the R M S error has been substantially reduced. Note that query based learning improves network training and the prediction capability.

(Refer Slide Time: 54:51)



Summary

In this lecture following topics have been covered

- RBF network model of Robot manipulators
- Dimensionally insufficient data
- Query based learning
- Model validation through network inversion

Finally summary; in this lecture following topics have been covered: radial basis function network model of robot manipulator, how do we model a robot manipulator radial basis function network? The problem was the dimensionally insufficient data because, the robot manipulator is open loop unstable system; hence the state independent control actuation is very difficult to actuate.

This specific problem has been sorted out to a very great extent by using the new noble concept called query based learning and the model validation through network inversion - is a very key concept that we learnt today in this class. Thank you very much.