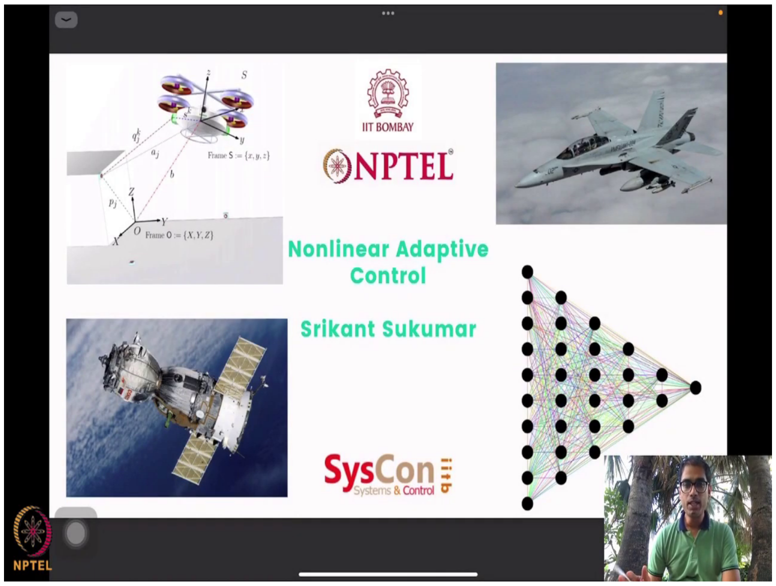
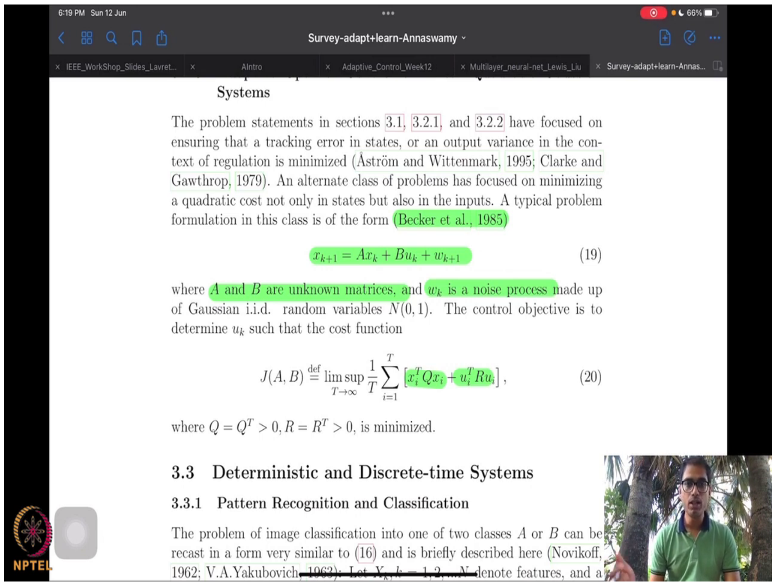
Nonlinear Adaptive Control
Professor Srikant Sukumar
Systems and Control
Indian Institute of Technology, Bombay
Week 12
Lecture No: 67
Real Time Neural Network Based Control of a Robotic Manipulator (Part 1)

(Refer Slide Time: 00:16)



Hello, everyone. Welcome to yet another session of our NPTEL on Nonlinear and Adaptive Control. I am Srikant Sukumar, Systems and Control, IIT, Bombay. So, we just started the 12th week of our lectures in this NPTEL course. So, this being the final week, I sort of started off with a little bit of history of how adaptive control evolved, and this is what we were looking at yesterday.

(Refer Slide Time: 00:51)


Systems

The problem statements in sections 3.1, 3.2.1, and 3.2.2 have focused on ensuring that a tracking error in states, or an output variance in the context of regulation is minimized (Åström and Wittenmark, 1995; Clarke and Gawthrop, 1979). An alternate class of problems has focused on minimizing a quadratic cost not only in states but also in the inputs. A typical problem formulation in this class is of the form (Becker et al., 1985)

$$x_{k+1} = Ax_k + Bu_k + w_{k+1} \quad (19)$$

where A and B are unknown matrices, and w_k is a noise process made up of Gaussian i.i.d. random variables $N(0,1)$. The control objective is to determine u_k such that the cost function

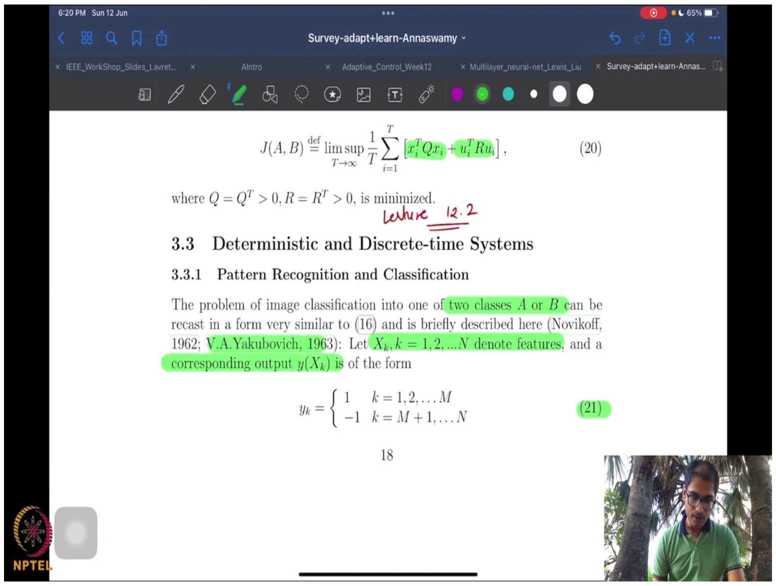
$$J(A,B) \overset{\text{def}}{=} \limsup_{T\to\infty} \frac{1}{T}\sum_{i=1}^{T}\left[x_i^T Q x_i + u_i^T R u_i\right], \quad (20)$$

where $Q = Q^T > 0, R = R^T > 0$, is minimized.

3.3 Deterministic and Discrete-time Systems

3.3.1 Pattern Recognition and Classification

The problem of image classification into one of two classes A or B can be recast in a form very similar to (16) and is briefly described here (Novikoff, 1962; V.A.Yakubovich, 1963): Let $X_k, k = 1,2,...N$ denote features, and a


$$J(A,B) \overset{\text{def}}{=} \limsup_{T\to\infty} \frac{1}{T}\sum_{i=1}^{T}\left[x_i^T Q x_i + u_i^T R u_i\right], \quad (20)$$

where $Q = Q^T > 0, R = R^T > 0$, is minimized.

Lecture 12.2

3.3 Deterministic and Discrete-time Systems

3.3.1 Pattern Recognition and Classification

The problem of image classification into one of two classes A or B can be recast in a form very similar to (16) and is briefly described here (Novikoff, 1962; V.A.Yakubovich, 1963): Let $X_k, k = 1,2,...N$ denote features, and a corresponding output $y(X_k)$ is of the form

$$y_k = \begin{cases} 1 & k = 1,2,\ldots M \\ -1 & k = M+1,\ldots N \end{cases} \quad (21)$$

18

So, if you, so, we were looking at this paper by Annaswamy. This is a paper that of course, I will post. And we are essentially looking at the chronology of how things evolved. We of course, saw a few things on how you started with the requirement for adaptation. Then you had this optimality-based MIT rule, and then there was a need for stable adaptive control.

Beyond that, you also had all the reference adaptive control, nonlinear adaptive control and epsilon sigma modifications parameter learning, but then there was also this parallel world where there was development for this stochastic discrete time systems, then there was a notion of learning there which was like, like neural networks, Adaline filters which involved parameter estimation again, or weight estimation again.

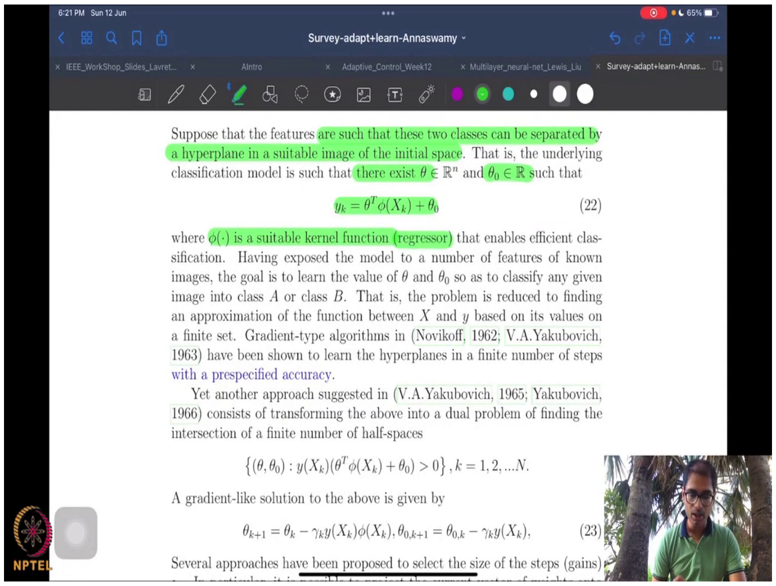
And then there was the reinforcement learning which involved the use of neural networks to estimate some value functions and so on. And, so overall there is, there is a lot of parallel developments where the notion of parameter learning was playing a rather big role. So, this was sort of what we have seen.

I quickly want to wrap up our discussion here on this paper. Of course, I will put this up, we have not even discussed half of the article, and very difficult to discuss it in entirety. But a lot of it are things beyond this, a lot of it are material that we have covered in this course, and so, it should not be very difficult for you to follow.

So, this is where I start my lecture 12.2. So, one of the problems that of course, started to be discussed in parallel with the evolution of adaptive control is the pattern recognition and classification problem. So, in this, what happens is that there are usually two classes A and B, and, and here there is basically some features which are denoted by X k.

And there is an output which is denoted by y k, and usually these outputs are in the form given in equation 21. So, this framework was of course in Yakubovic and Novikoff . So, they, what we want to do was to classify these images. And. So, the question that was sort of asked is, is there, is there such a, are such of these, possible to separate these classes by a hyperplane.

(Refer Slide Time: 03:38)



Suppose that the features are such that these two classes can be separated by a hyperplane in a suitable image of the initial space. That is, the underlying classification model is such that there exist $\theta \in \mathbb{R}^n$ and $\theta_0 \in \mathbb{R}$ such that

$$y_k = \theta^T \phi(X_k) + \theta_0 \quad (22)$$

where $\phi(\cdot)$ is a suitable kernel function (regressor) that enables efficient classification. Having exposed the model to a number of features of known images, the goal is to learn the value of θ and θ_0 so as to classify any given image into class A or class B. That is, the problem is reduced to finding an approximation of the function between X and y based on its values on a finite set. Gradient-type algorithms in (Novikoff, 1962; V.A.Yakubovich, 1963) have been shown to learn the hyperplanes in a finite number of steps with a prespecified accuracy.

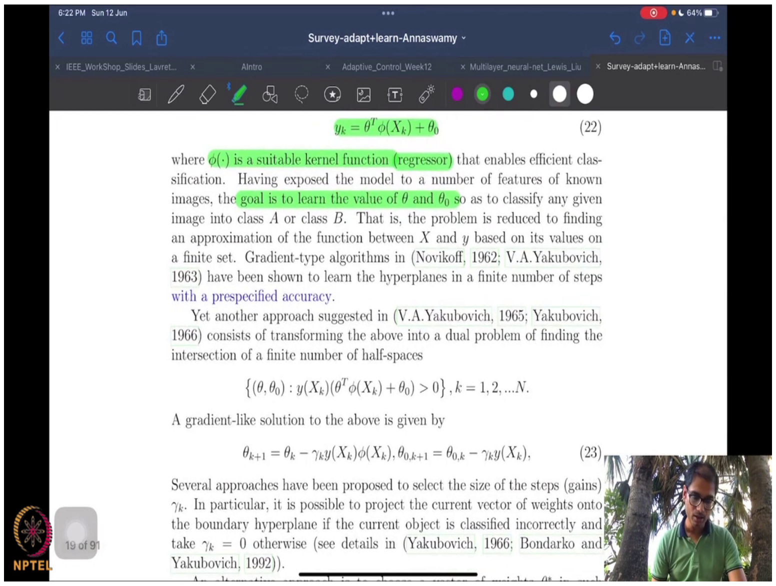
Yet another approach suggested in (V.A.Yakubovich, 1965; Yakubovich, 1966) consists of transforming the above into a dual problem of finding the intersection of a finite number of half-spaces

$$\left\{(\theta, \theta_0) : y(X_k)(\theta^T \phi(X_k) + \theta_0) > 0\right\}, k = 1, 2, ...N.$$

A gradient-like solution to the above is given by

$$\theta_{k+1} = \theta_k - \gamma_k y(X_k)\phi(X_k), \theta_{0,k+1} = \theta_{0,k} - \gamma_k y(X_k), \quad (23)$$

Several approaches have been proposed to select the size of the steps (gains)



$$y_k = \theta^T \phi(X_k) + \theta_0 \quad (22)$$

where $\phi(\cdot)$ is a suitable kernel function (regressor) that enables efficient classification. Having exposed the model to a number of features of known images, the goal is to learn the value of θ and θ_0 so as to classify any given image into class A or class B. That is, the problem is reduced to finding an approximation of the function between X and y based on its values on a finite set. Gradient-type algorithms in (Novikoff, 1962; V.A.Yakubovich, 1963) have been shown to learn the hyperplanes in a finite number of steps with a prespecified accuracy.

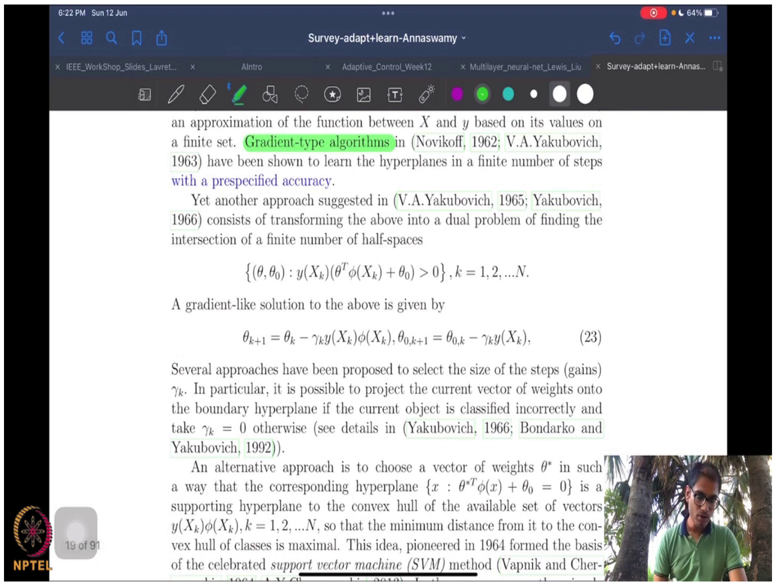
Yet another approach suggested in (V.A.Yakubovich, 1965; Yakubovich, 1966) consists of transforming the above into a dual problem of finding the intersection of a finite number of half-spaces

$$\left\{(\theta, \theta_0) : y(X_k)(\theta^T \phi(X_k) + \theta_0) > 0\right\}, k = 1, 2, ...N.$$

A gradient-like solution to the above is given by

$$\theta_{k+1} = \theta_k - \gamma_k y(X_k)\phi(X_k), \theta_{0,k+1} = \theta_{0,k} - \gamma_k y(X_k), \quad (23)$$

Several approaches have been proposed to select the size of the steps (gains)
γ_k. In particular, it is possible to project the current vector of weights onto the boundary hyperplane if the current object is classified incorrectly and take $\gamma_k = 0$ otherwise (see details in (Yakubovich, 1966; Bondarko and Yakubovich, 1992)).



an approximation of the function between X and y based on its values on a finite set. Gradient-type algorithms in (Novikoff, 1962; V.A.Yakubovich, 1963) have been shown to learn the hyperplanes in a finite number of steps with a prespecified accuracy.

Yet another approach suggested in (V.A.Yakubovich, 1965; Yakubovich, 1966) consists of transforming the above into a dual problem of finding the intersection of a finite number of half-spaces

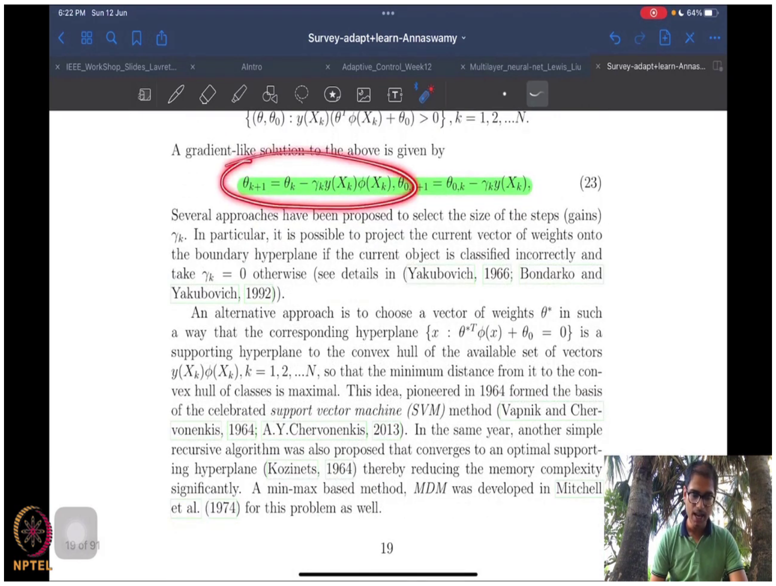
$$\left\{(\theta,\theta_0) : y(X_k)(\theta^T\phi(X_k)+\theta_0) > 0\right\}, k = 1, 2, ...N.$$

A gradient-like solution to the above is given by

$$\theta_{k+1} = \theta_k - \gamma_k y(X_k)\phi(X_k), \theta_{0,k+1} = \theta_{0,k} - \gamma_k y(X_k), \qquad (23)$$

Several approaches have been proposed to select the size of the steps (gains) γ_k. In particular, it is possible to project the current vector of weights onto the boundary hyperplane if the current object is classified incorrectly and take $\gamma_k = 0$ otherwise (see details in (Yakubovich, 1966; Bondarko and Yakubovich, 1992)).

An alternative approach is to choose a vector of weights θ^* in such a way that the corresponding hyperplane $\{x : \theta^{*T}\phi(x) + \theta_0 = 0\}$ is a supporting hyperplane to the convex hull of the available set of vectors $y(X_k)\phi(X_k), k = 1, 2, ...N$, so that the minimum distance from it to the convex hull of classes is maximal. This idea, pioneered in 1964 formed the basis of the celebrated *support vector machine (SVM)* method (Vapnik and Cher-



$$\left\{(\theta,\theta_0) : y(X_k)(\theta^T\phi(X_k)+\theta_0) > 0\right\}, k = 1, 2, ...N.$$

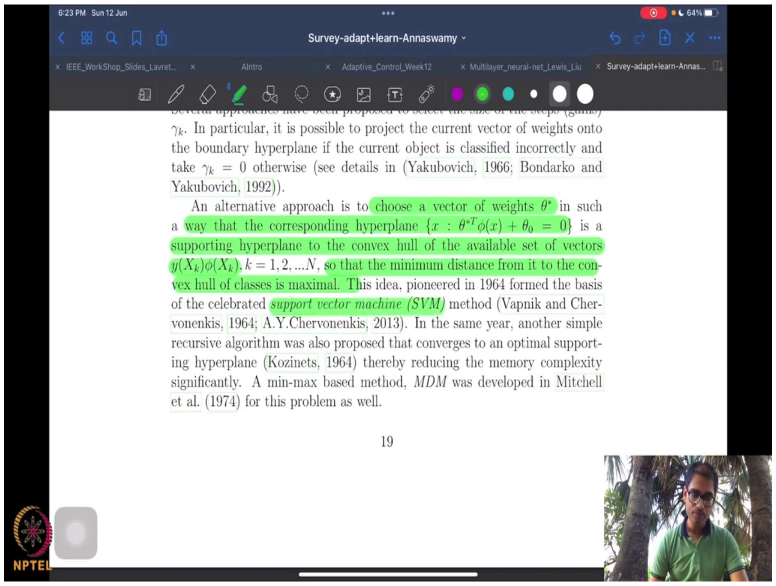
A gradient-like solution to the above is given by

$$\theta_{k+1} = \theta_k - \gamma_k y(X_k)\phi(X_k), \theta_{0,k+1} = \theta_{0,k} - \gamma_k y(X_k), \qquad (23)$$

Several approaches have been proposed to select the size of the steps (gains) γ_k. In particular, it is possible to project the current vector of weights onto the boundary hyperplane if the current object is classified incorrectly and take $\gamma_k = 0$ otherwise (see details in (Yakubovich, 1966; Bondarko and Yakubovich, 1992)).

An alternative approach is to choose a vector of weights θ^* in such a way that the corresponding hyperplane $\{x : \theta^{*T}\phi(x) + \theta_0 = 0\}$ is a supporting hyperplane to the convex hull of the available set of vectors $y(X_k)\phi(X_k), k = 1, 2, ...N$, so that the minimum distance from it to the convex hull of classes is maximal. This idea, pioneered in 1964 formed the basis of the celebrated *support vector machine (SVM)* method (Vapnik and Cher-vonenkis, 1964; A.Y.Chervonenkis, 2013). In the same year, another simple recursive algorithm was also proposed that converges to an optimal supporting hyperplane (Kozinets, 1964) thereby reducing the memory complexity significantly. A min-max based method, *MDM* was developed in Mitchell et al. (1974) for this problem as well.

19


Survey-adapt+learn-Annaswamy

γ_k. In particular, it is possible to project the current vector of weights onto the boundary hyperplane if the current object is classified incorrectly and take $\gamma_k = 0$ otherwise (see details in (Yakubovich, 1966; Bondarko and Yakubovich, 1992)).

An alternative approach is to choose a vector of weights θ^* in such a way that the corresponding hyperplane $\{x : \theta^{*T}\phi(x) + \theta_0 = 0\}$ is a supporting hyperplane to the convex hull of the available set of vectors $y(X_k)\phi(X_k), k = 1, 2, ...N$, so that the minimum distance from it to the convex hull of classes is maximal. This idea, pioneered in 1964 formed the basis of the celebrated *support vector machine (SVM)* method (Vapnik and Chervonenkis, 1964; A.Y.Chervonenkis, 2013). In the same year, another simple recursive algorithm was also proposed that converges to an optimal supporting hyperplane (Kozinets, 1964) thereby reducing the memory complexity significantly. A min-max based method, *MDM* was developed in Mitchell et al. (1974) for this problem as well.

19

And. So, the question that was sort of asked is, is there, is there such a, are such of these, possible to separate these classes by a hyperplane. And what does it mean to classify objects or these images into a hyperplane, means asking the question of whether there exists some parameter theta and theta naught such that you can write this output in this linear relationship with the parameters. So, this is this, and of course this phi is some suitable kernel function or regressor, in our adaptive control frame.

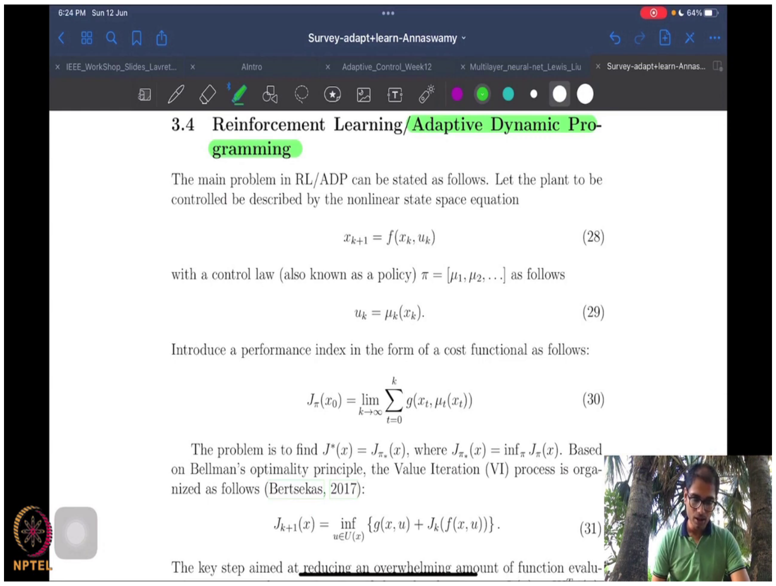
So, how this works is you, you try to expose the model to a lot of features and the goal is to learn this value theta, theta 0 so that you can classify any further images after the learning process into classes A and classes B. So, and of course, I mean how you separate them into classes, of course, I mean you look at this value right here. I mean whether you get, if you get 1, you are in class A, say, and if you get minus 1, you are in class B. So, that is how you do this classification problem. So, this is the image classification problem. But then, in recent times we would have seen sort of data classification problems also.

So, the interesting thing is now, again this problem is reduced to identifying some parameters and of course you can use this gradient type algorithms in order to find the values for these parameters, pre-specified accuracy of course. And then of course there are many gradient-type solutions such as the one you see here, which is, you will see that it is not very different from your adaptive feedback law or adaptive update law theorem.

This is just in discrete time, but has a very similar feel to your adaptive update law. So, so of course there is this, I mean, this, this sort of interesting idea, of course, there is an alternative approach where you choose some vector of weight such that this corresponding hyperplane is a supporting hyperplane to the convex hull of all these vectors.

So, so you sort of construct these vectors y X k phi X k, and then you sort of find these hyperplanes that are supporting hyperplanes to the convex hull of these vectors. So, so then, and of course so, that the minimum distance from it to the convex hull is maximum. So, so this is sort of how the support vector machine methods were, come about. But then you also see that there is this alternative method that we discussed first. And these are sort of closer to what we do in adaptive control.

(Refer Slide Time: 06:52)


6:24 PM Sun 12 Jun
64%
Survey-adapt+learn-Annaswamy
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewis_Liu
Survey-adapt+learn-Annas...
NPTEL

3.4 Reinforcement Learning/Adaptive Dynamic Programming

The main problem in RL/ADP can be stated as follows. Let the plant to be controlled be described by the nonlinear state space equation

$$x_{k+1} = f(x_k, u_k) \tag{28}$$

with a control law (also known as a policy) $\pi = [\mu_1, \mu_2, \ldots]$ as follows

$$u_k = \mu_k(x_k). \tag{29}$$

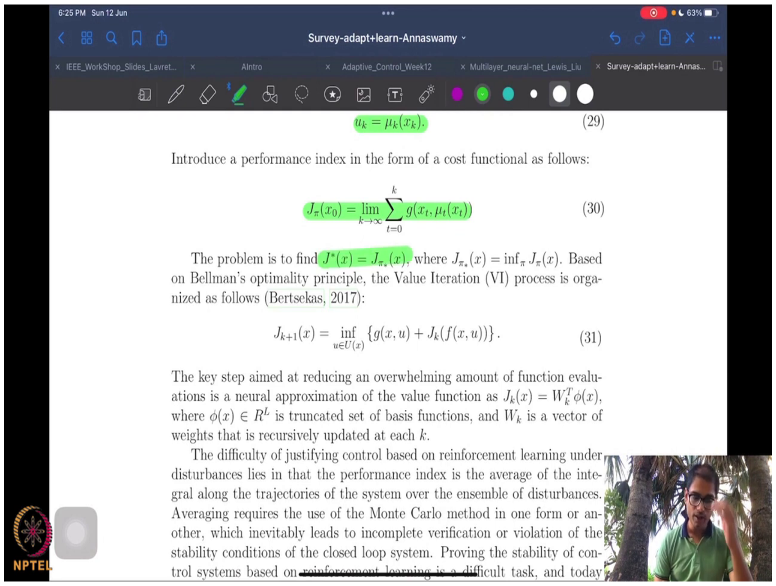
Introduce a performance index in the form of a cost functional as follows:

$$J_\pi(x_0) = \lim_{k\to\infty} \sum_{t=0}^{k} g(x_t, \mu_t(x_t)) \tag{30}$$

The problem is to find $J^*(x) = J_{\pi_*}(x)$, where $J_{\pi_*}(x) = \inf_\pi J_\pi(x)$. Based on Bellman's optimality principle, the Value Iteration (VI) process is organized as follows (Bertsekas, 2017):

$$J_{k+1}(x) = \inf_{u\in U(x)} \{g(x,u) + J_k(f(x,u))\}. \tag{31}$$

The key step aimed at reducing an overwhelming amount of function evalu-


6:25 PM Sun 12 Jun
Survey-adapt+learn-Annaswamy
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewis_Liu
Survey-adapt+learn-Annas...
NPTEL

$$u_k = \mu_k(x_k). \tag{29}$$

Introduce a performance index in the form of a cost functional as follows:

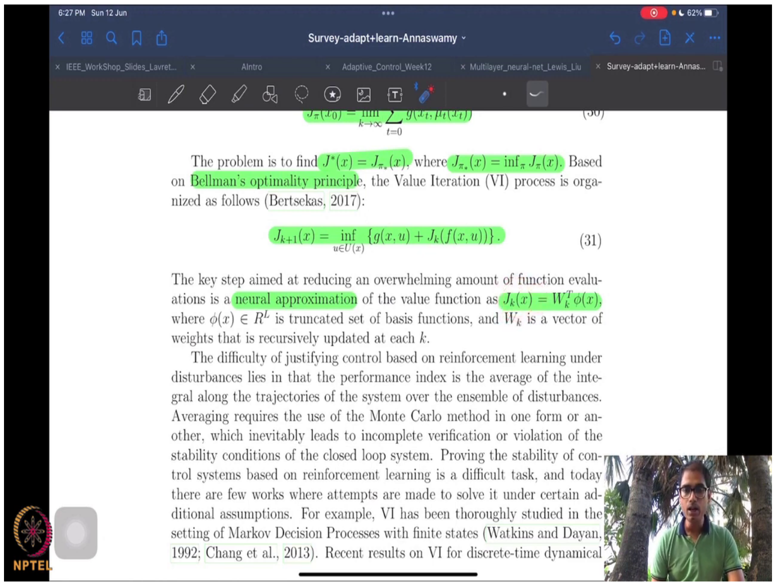
$$J_\pi(x_0) = \lim_{k\to\infty} \sum_{t=0}^{k} g(x_t, \mu_t(x_t)) \tag{30}$$

The problem is to find $J^*(x) = J_{\pi_*}(x)$, where $J_{\pi_*}(x) = \inf_\pi J_\pi(x)$. Based on Bellman's optimality principle, the Value Iteration (VI) process is organized as follows (Bertsekas, 2017):

$$J_{k+1}(x) = \inf_{u\in U(x)} \{g(x,u) + J_k(f(x,u))\}. \tag{31}$$

The key step aimed at reducing an overwhelming amount of function evaluations is a neural approximation of the value function as $J_k(x) = W_k^T\phi(x)$, where $\phi(x) \in R^L$ is truncated set of basis functions, and W_k is a vector of weights that is recursively updated at each k.

The difficulty of justifying control based on reinforcement learning under disturbances lies in that the performance index is the average of the integral along the trajectories of the system over the ensemble of disturbances. Averaging requires the use of the Monte Carlo method in one form or another, which inevitably leads to incomplete verification or violation of the stability conditions of the closed loop system. Proving the stability of control systems based on reinforcement learning is a difficult task, and today

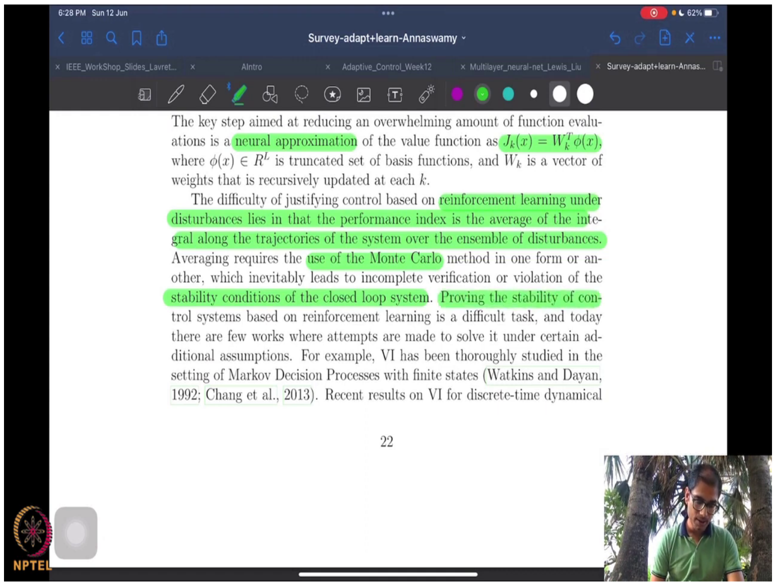

6:27 PM Sun 12 Jun
Survey-adapt+learn-Annaswamy
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewis_Liu
Survey-adapt+learn-Annas...
NPTEL

The problem is to find $J^*(x) = J_{\pi_*}(x)$, where $J_{\pi_*}(x) = \inf_\pi J_\pi(x)$. Based on Bellman's optimality principle, the Value Iteration (VI) process is organized as follows (Bertsekas, 2017):

$$J_{k+1}(x) = \inf_{u\in U(x)} \{g(x,u) + J_k(f(x,u))\}. \tag{31}$$

The key step aimed at reducing an overwhelming amount of function evaluations is a neural approximation of the value function as $J_k(x) = W_k^T\phi(x)$, where $\phi(x) \in R^L$ is truncated set of basis functions, and W_k is a vector of weights that is recursively updated at each k.

The difficulty of justifying control based on reinforcement learning under disturbances lies in that the performance index is the average of the integral along the trajectories of the system over the ensemble of disturbances. Averaging requires the use of the Monte Carlo method in one form or another, which inevitably leads to incomplete verification or violation of the stability conditions of the closed loop system. Proving the stability of control systems based on reinforcement learning is a difficult task, and today
there are few works where attempts are made to solve it under certain additional assumptions. For example, VI has been thoroughly studied in the setting of Markov Decision Processes with finite states (Watkins and Dayan, 1992; Chang et al., 2013). Recent results on VI for discrete-time dynamical


The key step aimed at reducing an overwhelming amount of function evaluations is a neural approximation of the value function as $J_k(x) = W_k^T \phi(x)$, where $\phi(x) \in R^L$ is truncated set of basis functions, and W_k is a vector of weights that is recursively updated at each k.

The difficulty of justifying control based on reinforcement learning under disturbances lies in that the performance index is the average of the integral along the trajectories of the system over the ensemble of disturbances. Averaging requires the use of the Monte Carlo method in one form or another, which inevitably leads to incomplete verification or violation of the stability conditions of the closed loop system. Proving the stability of control systems based on reinforcement learning is a difficult task, and today there are few works where attempts are made to solve it under certain additional assumptions. For example, VI has been thoroughly studied in the setting of Markov Decision Processes with finite states (Watkins and Dayan, 1992; Chang et al., 2013). Recent results on VI for discrete-time dynamical

22

So, anyway, so, then there was of course also the notion of reinforcement learning or adaptive dynamic programming. So, I am going to highlight this, that the fact that reinforcement learning is essentially adaptive dynamic program. And why is that? The idea is that you have some non-linear evolution and then you have what is called a control law which is called a policy in this discrete dynamic programming framework, and policies, essentially you have different control values at different instance of time, mu 1, mu 2 and so on.

And then, there is this notion of a performance index that you want to minimize. This performance index is like an infinite series or an infinite sum of this value function g. So, state pi is called a cost function. And the idea is to find the optimum, I want to find the optimal, and where the optimum, if you see that it is, you can see that it is indexed with a pi, and this pi is this policy.

So, it is an optimal with respect to the policy. So, you find the control law, if you will, or a control policy, such that this infinite sum is minimum. So, you have to choose. So, it is not, it is not just one choice of controller in discrete time. It is actually infinitely many choices. So, you have mu 1, mu 2, mu 3 all the way to negative infinity because you can see that this is an infinite sum.

So, so then in order to sort of look at this problem, how dynamic programming does it, and of course, this is, this is the expression for J pi star. You can see that it is infimum with respect to the control policy pi. So, using the Bellman's optimality principle, what you can, the value iteration process with this essentially organized like this.

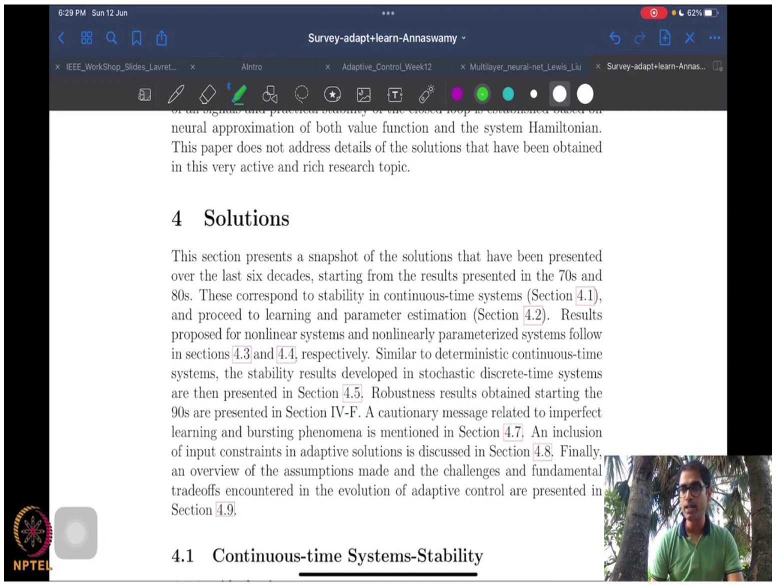
So, the optimal, how you yield this is that the optimal at the k plus 1th time is simply the infimum overall control values admissible, admissible control value. So, u x is the control, admissible control set. So, the admissible control means that suppose your control is to be bounded between, absolute value of control has to be less than 1, so that would be this set.

So, so you are essentially looking at the infimum over all control values of the function g and the optimal cost at the kth instant add it to it. So, this is how you sort of progress from one step to another in order to find the optimal control for that step. So, this is the optimal value of u for that step. So, of course to solve this, there is very, very large number of function computations to be done and. So, the idea here is to use a neural approximation for the function J k, and that essentially is approximated using as always a regressor phi and a set of parameters W k. So, you again see that there is a set of unknown vectors W k, which are weights, in this case positive weights, and then there are phis which are some truncated basis functions.

And these are used to approximate the value of this function. This is what is the basis for doing reinforcement learning. And, and this, identifying these weights is what is the adaptive part, this is the connection with adaptive control because you see that again I have some parameters that need to be learned. I mean, you can call them weights, but eventually they are parameters that need to be valued.

The only problem here is that if there are disturbances, then talking about stability is rather difficult because you, you sort of have to, if you want to look at stability and evolution of trajectories, you have to integrate over all such trajectories and this has to be some kind of Monte Carlo method that has to be used. And therefore, it is very difficult to, I mean come up with stability conditions for the closed system. I mean there are of course some results out there that talk about these but that is of course not a part of this article that we see here.

(Refer Slide Time: 11:27)


Survey-adapt+learn-Annaswamy
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewis_Liu
Survey-adapt+learn-Annas...
NPTEL

neural approximation of both value function and the system Hamiltonian. This paper does not address details of the solutions that have been obtained in this very active and rich research topic.

4 Solutions

This section presents a snapshot of the solutions that have been presented over the last six decades, starting from the results presented in the 70s and 80s. These correspond to stability in continuous-time systems (Section 4.1), and proceed to learning and parameter estimation (Section 4.2). Results proposed for nonlinear systems and nonlinearly parameterized systems follow in sections 4.3 and 4.4, respectively. Similar to deterministic continuous-time systems, the stability results developed in stochastic discrete-time systems are then presented in Section 4.5. Robustness results obtained starting the 90s are presented in Section IV-F. A cautionary message related to imperfect learning and bursting phenomena is mentioned in Section 4.7. An inclusion of input constraints in adaptive solutions is discussed in Section 4.8. Finally, an overview of the assumptions made and the challenges and fundamental tradeoffs encountered in the evolution of adaptive control are presented in Section 4.9.

4.1 Continuous-time Systems-Stability

So, anyway, so this is sort of where we stop our little bit of history lesson. Subsequently, there are many other sections in this paper on solutions in adaptive control and so on, and so lot of other interesting topics that have been covered in adaptive control. So, it gives a little bit of a flavor of everything there is in adaptive control. Some of this, of course, we have covered in our curriculum here, and some of it, we have not. So, I would strongly encourage you to look at this article to get a good idea of where we stand in terms of adaptive control and how it is connected to learning.

(Refer Slide Time: 12:40)

6:30 PM Sun 12 Jun
Multilayer_neural-net_Lewis_Liu
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewi...
Survey-adapt+learn-Annaswamy
NPTEL

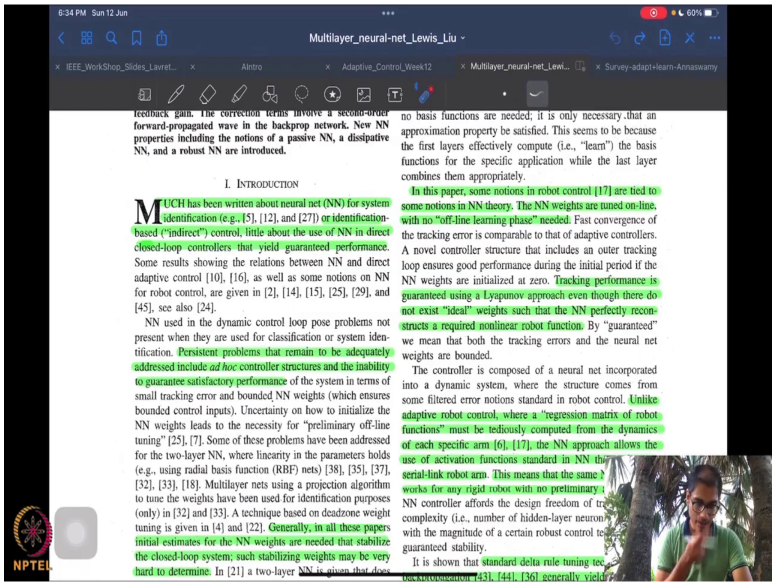
Multilayer Neural-Net Robot Controller with Guaranteed Tracking Performance

Frank L. Lewis, *Fellow, IEEE,* Aydin Yeşildirek, and Kai Liu, *Senior Member, IEEE*

***Abstract*—A multilayer neural-net (NN) controller for a general serial-link rigid robot arm is developed. The structure of the NN controller is derived using a filtered error/passivity approach. No off-line learning phase is needed for the proposed NN controller and the weights are easily initialized. The nonlinear nature of the NN, plus NN functional reconstruction inaccuracies and robot disturbances, mean that the standard delta rule using backpropagation tuning does not suffice for closed-loop dynamic control. Novel on-line weight tuning algorithms, including correction terms to the delta rule plus an added robustifying signal, guarantee bounded tracking errors as well as bounded NN weights. Specific bounds are determined, and the tracking error bound can be made arbitrarily small by increasing a certain feedback gain. The correction terms involve a second-order forward-propagated wave in the backprop network. New NN properties including the notions of a passive NN, a dissipative NN, and a robust NN are introduced.**

not need such initial estimates; in that work, the NN is linear in the tunable weights so that suitable basis functions must be first selected (e.g., RBF's).

In this paper we extend work in [19]–[21] to confront these deficiencies for the full nonlinear three-layer NN with arbitrary activation functions (as long as the function satisfies an approximation property and it and its derivatives a
The approximation accuracy is generally better
multilayer NN than in linear two-layer NN. N
comparison to the linear-in-the-parameter NN u
[35], [37], [32], [33], [18], and [21], in the mu
no basis functions are needed; it is only neces
approximation property be satisfied. This seems t
the first layers effectively compute (i.e., "learn
functions for the specific application while th


6:34 PM Sun 12 Jun
Multilayer_neural-net_Lewis_Liu
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewi...
Survey-adapt+learn-Annaswamy
NPTEL

feedback gain. The correction terms involve a second-order forward-propagated wave in the backprop network. New NN properties including the notions of a passive NN, a dissipative NN, and a robust NN are introduced.

I. Introduction

MUCH has been written about neural net (NN) for system identification (e.g., [5], [12], and [27]) or identification-based ("indirect") control, little about the use of NN in direct closed-loop controllers that yield guaranteed performance. Some results showing the relations between NN and direct adaptive control [10], [16], as well as some notions on NN for robot control, are given in [2], [14], [15], [25], [29], and [45], see also [24].

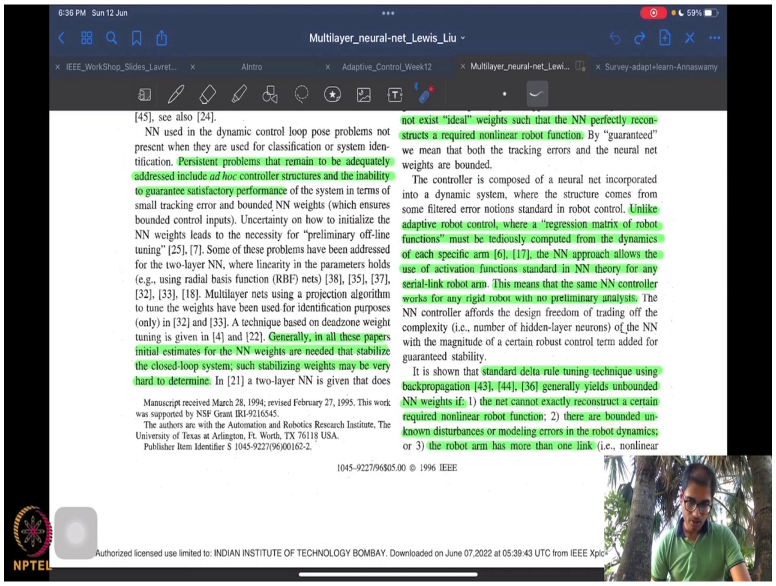
NN used in the dynamic control loop pose problems not present when they are used for classification or system identification. Persistent problems that remain to be adequately addressed include *ad hoc* controller structures and the inability to guarantee satisfactory performance of the system in terms of small tracking error and bounded NN weights (which ensures bounded control inputs). Uncertainty on how to initialize the NN weights leads to the necessity for "preliminary off-line tuning" [25], [7]. Some of these problems have been addressed for the two-layer NN, where linearity in the parameters holds (e.g., using radial basis function (RBF) nets) [38], [35], [37], [32], [33], [18]. Multilayer nets using a projection algorithm to tune the weights have been used for identification purposes (only) in [32] and [33]. A technique based on deadzone weight tuning is given in [4] and [22]. Generally, in all these papers initial estimates for the NN weights are needed that stabilize the closed-loop system; such stabilizing weights may be very hard to determine. In [21] a two-layer NN is given that does

no basis functions are needed; it is only necessary that an approximation property be satisfied. This seems to be because the first layers effectively compute (i.e., "learn") the basis functions for the specific application while the last layer combines them appropriately.

In this paper, some notions in robot control [17] are tied to some notions in NN theory. The NN weights are tuned on-line, with no "off-line learning phase" needed. Fast convergence of the tracking error is comparable to that of adaptive controllers. A novel controller structure that includes an outer tracking loop ensures good performance during the initial period if the NN weights are initialized at zero. Tracking performance is guaranteed using a Lyapunov approach even though there do not exist "ideal" weights such that the NN perfectly reconstructs a required nonlinear robot function. By "guaranteed" we mean that both the tracking errors and the neural net weights are bounded.

The controller is composed of a neural net incorporated into a dynamic system, where the structure comes from some filtered error notions standard in robot control. Unlike adaptive robot control, where a "regression matrix of robot functions" must be tediously computed from the dynamics of each specific arm [6], [17], the NN approach allows the use of activation functions standard in NN th
serial-link robot arm. This means that the same N
works for any rigid robot with no preliminary
NN controller affords the design freedom of tr
complexity (i.e., number of hidden-layer neuron
with the magnitude of a certain robust control te
guaranteed stability.

It is shown that standard delta rule tuning tec
backpropagation [43], [44], [36] generally yield



[45], see also [24].

NN used in the dynamic control loop pose problems not present when they are used for classification or system identification. Persistent problems that remain to be adequately addressed include *ad hoc* controller structures and the inability to guarantee satisfactory performance of the system in terms of small tracking error and bounded NN weights (which ensures bounded control inputs). Uncertainty on how to initialize the NN weights leads to the necessity for "preliminary off-line tuning" [25], [7]. Some of these problems have been addressed for the two-layer NN, where linearity in the parameters holds (e.g., using radial basis function (RBF) nets) [38], [35], [37], [32], [33], [18]. Multilayer nets using a projection algorithm to tune the weights have been used for identification purposes (only) in [32] and [33]. A technique based on deadzone weight tuning is given in [4] and [22]. Generally, in all these papers initial estimates for the NN weights are needed that stabilize the closed-loop system; such stabilizing weights may be very hard to determine. In [21] a two-layer NN is given that does not exist "ideal" weights such that the NN perfectly reconstructs a required nonlinear robot function. By "guaranteed" we mean that both the tracking errors and the neural net weights are bounded.

The controller is composed of a neural net incorporated into a dynamic system, where the structure comes from some filtered error notions standard in robot control. Unlike adaptive robot control, where a "regression matrix of robot functions" must be tediously computed from the dynamics of each specific arm [6], [17], the NN approach allows the use of activation functions standard in NN theory for any serial-link robot arm. This means that the same NN controller works for any rigid robot with no preliminary analysis. The NN controller affords the design freedom of trading off the complexity (i.e., number of hidden-layer neurons) of the NN with the magnitude of a certain robust control term added for guaranteed stability.

It is shown that standard delta rule tuning technique using backpropagation [43], [44], [36] generally yields unbounded NN weights if: 1) the net cannot exactly reconstruct a certain required nonlinear robot function; 2) there are bounded unknown disturbances or modeling errors in the robot dynamics; or 3) the robot arm has more than one link (i.e., nonlinear

Manuscript received March 28, 1994; revised February 27, 1995. This work was supported by NSF Grant IRI-9216545.

The authors are with the Automation and Robotics Research Institute, The University of Texas at Arlington, Ft. Worth, TX 76118 USA.

Publisher Item Identifier S 1045-9227(96)00162-2.

1045-9227/96$05.00 © 1996 IEEE

Authorized licensed use limited to: INDIAN INSTITUTE OF TECHNOLOGY BOMBAY. Downloaded on June 07,2022 at 05:39:43 UTC from IEEE Xplore.

6:36 PM Sun 12 Jun
Multilayer_neural-net_Lewis_Liu
IEEE_WorkShop_Slides_Lavret...
AIntro
Adaptive_Control_Week12
Multilayer_neural-net_Lewi...
Survey-adapt+learn-Annaswamy
59%
NPTEL

What I want to do now is to again go to another article, I do not know why I have used this effect, I apologize. So, I want to sort of go here, to this article, and this is an article, it is from 1996 from Frank Lewis and co-authors. And this article actually talks about implementation of an adaptive controller via multi-layered neural network or an adaptive controller via multi-layered neural network.

And this is being used as a robot. So, this is one of the, probably one of the first multi-layered implementations of neural network, adaptive control in neural networks with nice tracking stability performance results, which is, so, therefore I sort of wanted to look at this article individually in this last week so that you can get an idea of how actually stability guarantees in neural networks can be provided.

So, again you see that this is an old article, and this is a rather specific article, meaning, you look at real real-time application of neural network. In a lot of literature that you see now or a lot of applications that you look at now, most learning algorithms are implement are offline implementations in the sense that the entire training of the algorithm happens offline.

But this is one of the few results that you will see, especially connected to control, where you have online learning of the adaptive control also. Now, one can of course question the advantage and disadvantage either. Of course, it is true that if you do offline training, you can take in a lot more data and you can do much better learning of parameters and learn probably for a larger variety of cases and so on.

But one of the problems is that with, this offline learning and then implementation online is that if your initial conditions and your of your dynamical system and your operational conditions of your dynamical system changes a lot, then for example, if I, if I did a lot of, I wanted to look at a sector and I want to fly my unmanned vehicle using some images, image processing, and I made a map for example in, in dawn, at dawn time.

So I did all my training at early morning and I used images that I obtained using an early morning flight, a bunch of early flights using this UAV, but then suppose I fly my, actually, I actually find my UAV in the evening time where the light is a little bit less or maybe in the daytime where the light is a little bit more, then the results become significantly different.

So, if the initial conditions, operational conditions change from the training data, then your performance can significantly deteriorate. On the other hand, if you are actually using some kind of an online learning of this neural network, then you can continue to learn even when things change, and you do not deteriorate a performance significantly.

So, that is one of the advantages, but of course the limitation is that you can only have a few layers for computational efficiency, and, in order to be able to implement things in real time. So, that is the idea. We are looking at a real-time implementation of a multi-layered neural network, and we do this in the adaptive control framework that we have seen.

So, so a lot of results have been written about this adaptive neural network and, and but there is little about the use of neural network in direct closed loop controllers, at least until 96. There is a little bit more now, but again, not a lot. This is still an open area of research, especially the part about tuning stability.

So, one of the issues is, is sort of that need to be addressed adequately is the inclusion of ad hoc controller structures and the inability to guarantee satisfactory performance. So, so one of the things that is required in a lot of articles that these authors mentioned is that you need initial estimates for the neural network weights in order for performance to be good.

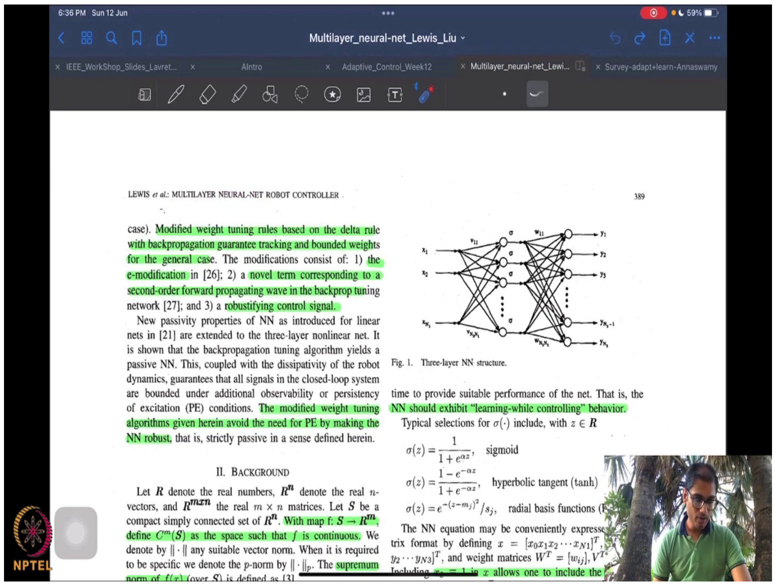
And identifying such stabilizing weights may not be very easy before you even start running the system. So, so the idea in this article, as the authors say, is to confront these deficiency for full nonlinear three-layer neural network with arbitrary activation functions. Of course, there is concepts of robot control, we have discussed here a little bit because the application is a robotic manipulator.

And the tracking performance is guaranteed using Lyapunov approach Like, remember there is no ideal weights that are identified. And typically, when you use a standard adaptive controller for robot, your regress, you regressor matrix, the regressor matrix that is used in all these, needs to be very previously computed for every specific robot.

For example, if I go from a three-link robot to five-link robot or you go from a linear joint to a articulate or a rotary joint, things change significantly. So, you have to compute the regressor matrix very carefully in order to apply an adaptive controller. But the advantage of using a neural network based adaptive control is that it will automatically learn these parameters. So, these, using these activation functions.

So, it can be applied for any serial link robotic arm. So, this is the advantage. If one controller works for any serial link robotic arm just like a package. So, one of the things that is sort of demonstrated in this article is that the standard tuning using backpropagation-based methods, yields, can yield unbounded neural network weights. And especially when there is disturbances and, the robot arm is more than one link, which is sort of a very, very basic requirement that the robot may have more than one link.

(Refer Slide Time: 19:28)



6:36 PM Sun 12 Jun

Multilayer_neural-net_Lewis_Liu

LEWIS *et al.*: MULTILAYER NEURAL-NET ROBOT CONTROLLER 389

case). Modified weight tuning rules based on the delta rule with backpropagation guarantee tracking and bounded weights for the general case. The modifications consist of: 1) the e-modification in [26]; 2) a novel term corresponding to a second-order forward propagating wave in the backprop tuning network [27]; and 3) a robustifying control signal.

New passivity properties of NN as introduced for linear nets in [21] are extended to the three-layer nonlinear net. It is shown that the backpropagation tuning algorithm yields a passive NN. This, coupled with the dissipativity of the robot dynamics, guarantees that all signals in the closed-loop system are bounded under additional observability or persistency of excitation (PE) conditions. The modified weight tuning algorithms given herein avoid the need for PE by making the NN robust, that is, strictly passive in a sense defined herein.

II. BACKGROUND

Let $\boldsymbol{R}$ denote the real numbers, $\boldsymbol{R}^n$ denote the real n-vectors, and $\boldsymbol{R}^{mxn}$ the real $m \times n$ matrices. Let $\boldsymbol{S}$ be a compact simply connected set of $\boldsymbol{R}^n$. With map f: $\boldsymbol{S} \rightarrow \boldsymbol{R}^m$, define $C^m(\boldsymbol{S})$ as the space such that f is continuous. We denote by $\|\cdot\|$ any suitable vector norm. When it is required to be specific we denote the p-norm by $\|\cdot\|_p$. The supremum

Fig. 1. Three-layer NN structure.

time to provide suitable performance of the net. That is, the NN should exhibit "learning-while controlling" behavior.

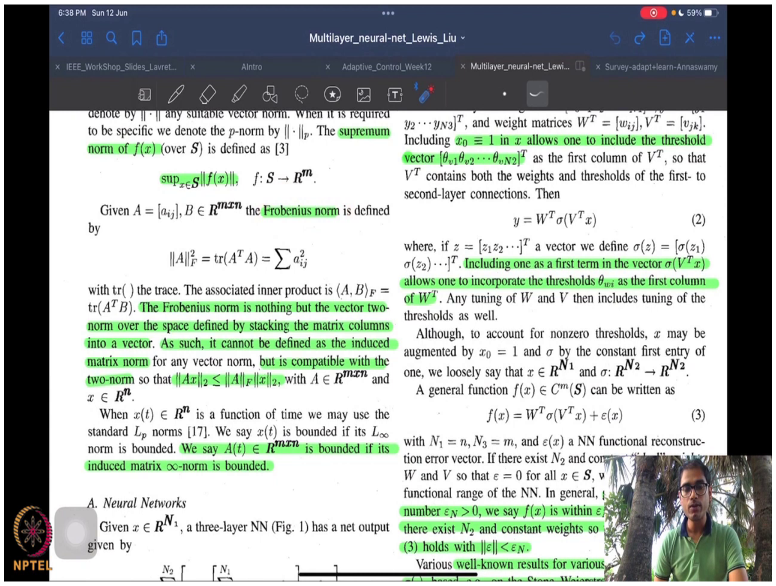
Typical selections for $\sigma(\cdot)$ include, with $z \in \boldsymbol{R}$

$$\sigma(z) = \frac{1}{1+e^{\alpha z}}, \quad \text{sigmoid}$$

$$\sigma(z) = \frac{1-e^{-\alpha z}}{1+e^{-\alpha z}}, \quad \text{hyperbolic tangent (tanh)}$$

$$\sigma(z) = e^{-(z-m_j)^2}/s_j, \quad \text{radial basis functions (}$$

The NN equation may be conveniently expresse trix format by defining $x = [x_0 x_1 x_2 \cdots x_{N1}]^T$, $y_2 \cdots y_{N3}]^T$, and weight matrices $W^T = [w_{ij}]$, V^T Including $x_0 \equiv 1$ in x allows one to include the



6:38 PM Sun 12 Jun

Multilayer_neural-net_Lewis_Liu

denote by $\|\cdot\|$ any suitable vector norm. When it is required to be specific we denote the p-norm by $\|\cdot\|_p$. The supremum norm of $f(x)$ (over $\boldsymbol{S}$) is defined as [3]

$$\sup_{x \in \boldsymbol{S}} \|f(x)\|, \quad f: \boldsymbol{S} \rightarrow \boldsymbol{R}^m.$$

Given $A = [a_{ij}], B \in \boldsymbol{R}^{mxn}$ the Frobenius norm is defined by

$$\|A\|_F^2 = \operatorname{tr}(A^T A) = \sum a_{ij}^2$$

with tr( ) the trace. The associated inner product is $\langle A, B\rangle_F = \operatorname{tr}(A^T B)$. The Frobenius norm is nothing but the vector two-norm over the space defined by stacking the matrix columns into a vector. As such, it cannot be defined as the induced matrix norm for any vector norm, but is compatible with the two-norm so that $\|Ax\|_2 \leq \|A\|_F \|x\|_2$, with $A \in \boldsymbol{R}^{mxn}$ and $x \in \boldsymbol{R}^n$.

When $x(t) \in \boldsymbol{R}^n$ is a function of time we may use the standard L_p norms [17]. We say $x(t)$ is bounded if its L_∞ norm is bounded. We say $A(t) \in \boldsymbol{R}^{mxn}$ is bounded if its induced matrix ∞-norm is bounded.

A. Neural Networks

Given $x \in \boldsymbol{R}^{N_1}$, a three-layer NN (Fig. 1) has a net output given by

$y_2 \cdots y_{N3}]^T$, and weight matrices $W^T = [w_{ij}]$, $V^T = [v_{jk}]$. Including $x_0 \equiv 1$ in x allows one to include the threshold vector $[\theta_{v1} \theta_{v2} \cdots \theta_{vN2}]^T$ as the first column of V^T, so that V^T contains both the weights and thresholds of the first- to second-layer connections. Then

$$y = W^T \sigma(V^T x) \qquad (2)$$

where, if $z = [z_1 z_2 \cdots]^T$ a vector we define $\sigma(z) = [\sigma(z_1) \; \sigma(z_2) \cdots]^T$. Including one as a first term in the vector $\sigma(V^T x)$ allows one to incorporate the thresholds θ_{wi} as the first column of W^T. Any tuning of W and V then includes tuning of the thresholds as well.

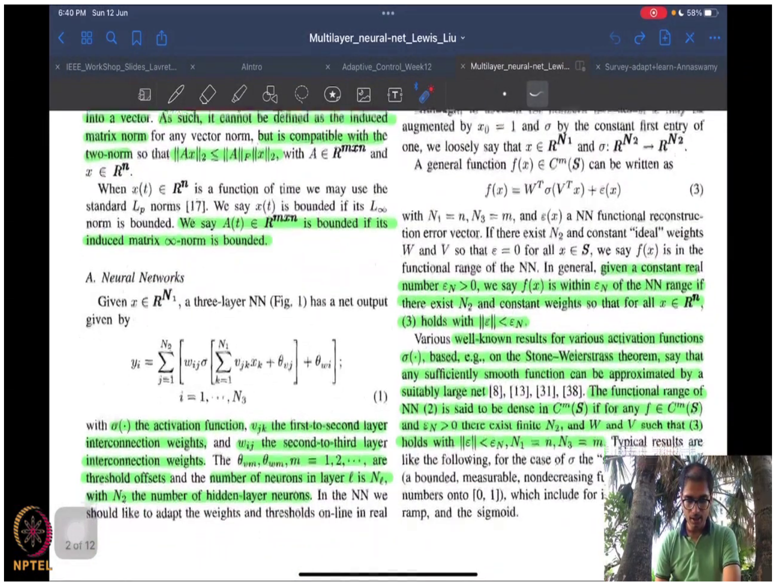
Although, to account for nonzero thresholds, x may be augmented by $x_0 = 1$ and σ by the constant first entry of one, we loosely say that $x \in \boldsymbol{R}^{N_1}$ and σ: $\boldsymbol{R}^{N_2} \rightarrow \boldsymbol{R}^{N_2}$.

A general function $f(x) \in C^m(\boldsymbol{S})$ can be written as

$$f(x) = W^T \sigma(V^T x) + \varepsilon(x) \qquad (3)$$

with $N_1 = n, N_3 = m$, and $\varepsilon(x)$ a NN functional reconstruction error vector. If there exist N_2 and con W and V so that $\varepsilon = 0$ for all $x \in \boldsymbol{S}$, w functional range of the NN. In general, number $\varepsilon_N > 0$, we say $f(x)$ is within ε there exist N_2 and constant weights so (3) holds with $\|\varepsilon\| < \varepsilon_N$.

Various well-known results for various



into a vector. As such, it cannot be defined as the induced matrix norm for any vector norm, but is compatible with the two-norm so that $\|Ax\|_2 \le \|A\|_F \|x\|_2$, with $A \in R^{mxn}$ and $x \in R^n$.

When $x(t) \in R^n$ is a function of time we may use the standard L_p norms [17]. We say $x(t)$ is bounded if its L_∞ norm is bounded. We say $A(t) \in R^{mxn}$ is bounded if its induced matrix ∞-norm is bounded.

A. Neural Networks

Given $x \in R^{N_1}$, a three-layer NN (Fig. 1) has a net output given by

$$y_i = \sum_{j=1}^{N_2} \left[w_{ij} \sigma \left[\sum_{k=1}^{N_1} v_{jk} x_k + \theta_{vj} \right] + \theta_{wi} \right]; \quad i = 1, \cdots, N_3 \tag{1}$$

with $\sigma(\cdot)$ the activation function, v_{jk} the first-to-second layer interconnection weights, and w_{ij} the second-to-third layer interconnection weights. The $\theta_{vm}, \theta_{wm}, m = 1, 2, \cdots,$ are threshold offsets and the number of neurons in layer ℓ is N_ℓ, with N_2 the number of hidden-layer neurons. In the NN we should like to adapt the weights and thresholds on-line in real

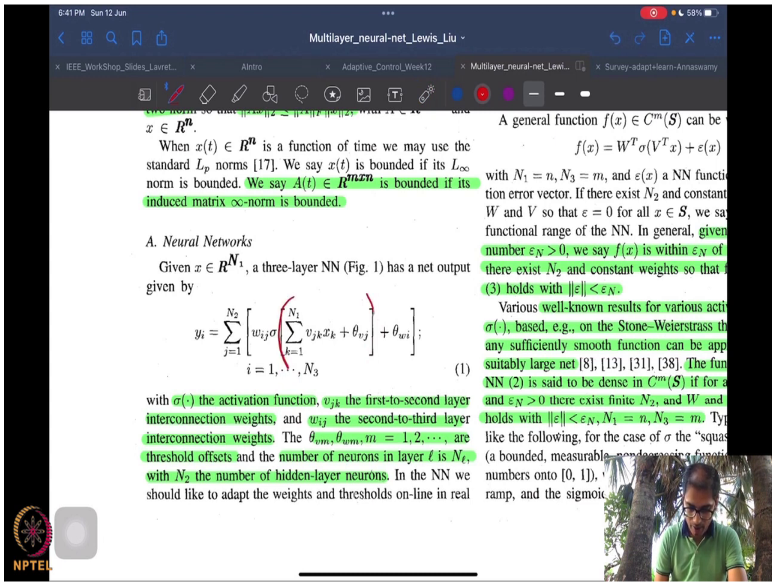
augmented by $x_0 = 1$ and σ by the constant first entry of one, we loosely say that $x \in R^{N_1}$ and σ: $R^{N_2} \to R^{N_2}$.

A general function $f(x) \in C^m(S)$ can be written as

$$f(x) = W^T \sigma(V^T x) + \varepsilon(x) \tag{3}$$

with $N_1 = n, N_3 = m$, and $\varepsilon(x)$ a NN functional reconstruction error vector. If there exist N_2 and constant "ideal" weights W and V so that $\varepsilon = 0$ for all $x \in S$, we say $f(x)$ is in the functional range of the NN. In general, given a constant real number $\varepsilon_N > 0$, we say $f(x)$ is within ε_N of the NN range if there exist N_2 and constant weights so that for all $x \in R^n$, (3) holds with $\|\varepsilon\| < \varepsilon_N$.

Various well-known results for various activation functions $\sigma(\cdot)$, based, e.g., on the Stone–Weierstrass theorem, say that any sufficiently smooth function can be approximated by a suitably large net [8], [13], [31], [38]. The functional range of NN (2) is said to be dense in $C^m(S)$ if for any $f \in C^m(S)$ and $\varepsilon_N > 0$ there exist finite N_2, and W and V such that (3) holds with $\|\varepsilon\| < \varepsilon_N, N_1 = n, N_3 = m$. Typical results are like the following, for the case of σ the "
(a bounded, measurable, nondecreasing fu
numbers onto [0, 1]), which include for i
ramp, and the sigmoid.



$x \in R^n$.

When $x(t) \in R^n$ is a function of time we may use the standard L_p norms [17]. We say $x(t)$ is bounded if its L_∞ norm is bounded. We say $A(t) \in R^{mxn}$ is bounded if its induced matrix ∞-norm is bounded.

A. Neural Networks

Given $x \in R^{N_1}$, a three-layer NN (Fig. 1) has a net output given by

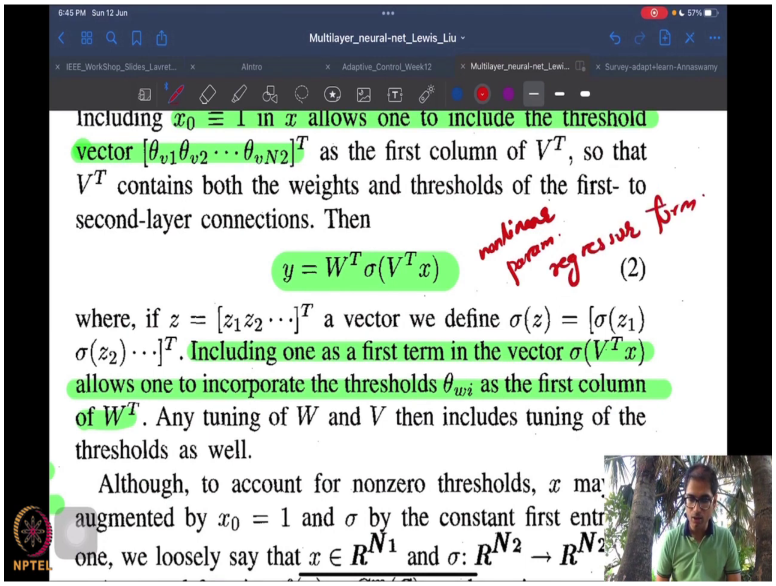
$$y_i = \sum_{j=1}^{N_2} \left[w_{ij} \sigma \left(\sum_{k=1}^{N_1} v_{jk} x_k + \theta_{vj} \right) + \theta_{wi} \right]; \quad i = 1, \cdots, N_3 \tag{1}$$

with $\sigma(\cdot)$ the activation function, v_{jk} the first-to-second layer interconnection weights, and w_{ij} the second-to-third layer interconnection weights. The $\theta_{vm}, \theta_{wm}, m = 1, 2, \cdots,$ are threshold offsets and the number of neurons in layer ℓ is N_ℓ, with N_2 the number of hidden-layer neurons. In the NN we should like to adapt the weights and thresholds on-line in real

A general function $f(x) \in C^m(S)$ can be

$f(x) = W^T \sigma(V^T x) + \varepsilon(x)$

with $N_1 = n, N_3 = m$, and $\varepsilon(x)$ a NN functi
tion error vector. If there exist N_2 and constant
W and V so that $\varepsilon = 0$ for all $x \in S$, we say
functional range of the NN. In general, given
number $\varepsilon_N > 0$, we say $f(x)$ is within ε_N of
there exist N_2 and constant weights so that f
(3) holds with $\|\varepsilon\| < \varepsilon_N$.
Various well-known results for various acti
$\sigma(\cdot)$, based, e.g., on the Stone–Weierstrass th
any sufficiently smooth function can be app
suitably large net [8], [13], [31], [38]. The fun
NN (2) is said to be dense in $C^m(S)$ if for a
and $\varepsilon_N > 0$ there exist finite N_2, and W and
holds with $\|\varepsilon\| < \varepsilon_N, N_1 = n, N_3 = m$. Typ
like the following, for the case of σ the "squa
(a bounded, measurable, nondecreasing functi
numbers onto [0, 1]),
ramp, and the sigmoi

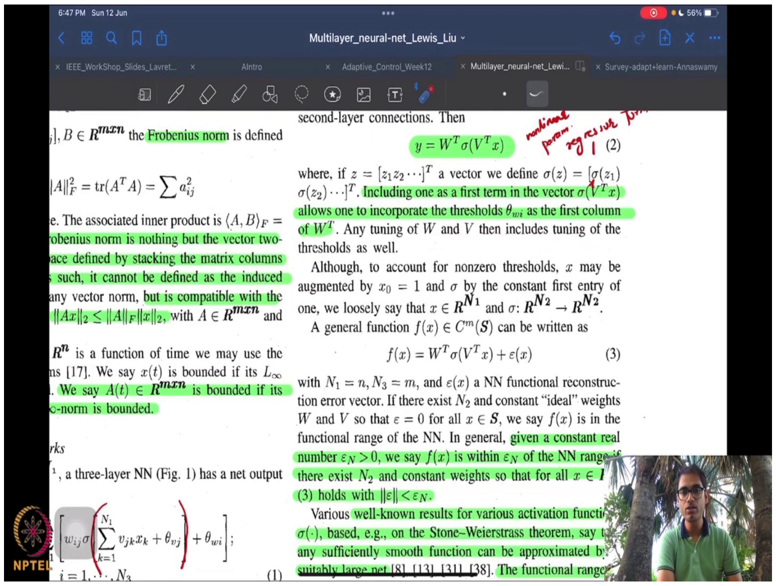

Including $x_0 \equiv 1$ in x allows one to include the threshold vector $[\theta_{v1}\theta_{v2}\cdots\theta_{vN2}]^T$ as the first column of V^T, so that V^T contains both the weights and thresholds of the first- to second-layer connections. Then

$$y = W^T\sigma(V^Tx) \quad (2)$$

nonlinear param. regressive form.

where, if $z = [z_1 z_2 \cdots]^T$ a vector we define $\sigma(z) = [\sigma(z_1)\ \sigma(z_2)\cdots]^T$. Including one as a first term in the vector $\sigma(V^Tx)$ allows one to incorporate the thresholds θ_{wi} as the first column of W^T. Any tuning of W and V then includes tuning of the thresholds as well.

Although, to account for nonzero thresholds, x ma
augmented by $x_0 = 1$ and σ by the constant first entr
one, we loosely say that $x \in \mathbf{R}^{N_1}$ and $\sigma\colon \mathbf{R}^{N_2} \rightarrow \mathbf{R}^{N_2}$


], $B \in \mathbf{R}^{mxn}$ the Frobenius norm is defined

$$\|A\|_F^2 = \mathrm{tr}(A^TA) = \sum a_{ij}^2$$

e. The associated inner product is $\langle A,B\rangle_F =$
obenius norm is nothing but the vector two-
ace defined by stacking the matrix columns
such, it cannot be defined as the induced
any vector norm, but is compatible with the
$\|Ax\|_2 \le \|A\|_F\|x\|_2$, with $A \in \mathbf{R}^{mxn}$ and

$\mathbf{R}^n$ is a function of time we may use the
ns [17]. We say $x(t)$ is bounded if its L_∞
. We say $A(t) \in \mathbf{R}^{mxn}$ is bounded if its
-norm is bounded.

rks

, a three-layer NN (Fig. 1) has a net output

$$\left[w_{ij}\sigma\left(\sum_{k=1}^{N_1} v_{jk}x_k + \theta_{vj}\right) + \theta_{wi}\right]; \quad i = 1,\ldots,N_3 \quad (1)$$

second-layer connections. Then

$$y = W^T\sigma(V^Tx) \quad (2)$$

nonlinear param. regressive form

where, if $z = [z_1 z_2 \cdots]^T$ a vector we define $\sigma(z) = [\sigma(z_1)\ \sigma(z_2)\cdots]^T$. Including one as a first term in the vector $\sigma(V^Tx)$ allows one to incorporate the thresholds θ_{wi} as the first column of W^T. Any tuning of W and V then includes tuning of the thresholds as well.

Although, to account for nonzero thresholds, x may be augmented by $x_0 = 1$ and σ by the constant first entry of one, we loosely say that $x \in \mathbf{R}^{N_1}$ and $\sigma\colon \mathbf{R}^{N_2} \rightarrow \mathbf{R}^{N_2}$.

A general function $f(x) \in C^m(\mathbf{S})$ can be written as

$$f(x) = W^T\sigma(V^Tx) + \varepsilon(x) \quad (3)$$

with $N_1 = n, N_3 = m$, and $\varepsilon(x)$ a NN functional reconstruction error vector. If there exist N_2 and constant "ideal" weights W and V so that $\varepsilon = 0$ for all $x \in \mathbf{S}$, we say $f(x)$ is in the functional range of the NN. In general, given a constant real number $\varepsilon_N > 0$, we say $f(x)$ is within ε_N of the NN range if there exist N_2 and constant weights so that for all $x \in$
(3) holds with $\|\varepsilon\| < \varepsilon_N$.

Various well-known results for various activation functi
$\sigma(\cdot)$, based, e.g., on the Stone–Weierstrass theorem, say t
any sufficiently smooth function can be approximated b
suitably large net [8], [13], [31], [38]. The functional range

So, there are modified weight tuning methods are also proposed here, which, which of course, selected the epsilon modifications. So, so the modified weight tuning approaches also, the authors claim that they can avoid the PE condition, which is also something rather, it is good, rather strong, rather a strong agent.

Now, so, let us sort of start looking into what the problem set up is. So, we know that we are doing learning using adaptive methods. So, typical neural network parameter estimation algorithms can rely on some kind of again, gradient descent type formula, some kind of optimization-based formula but here everything is based on adaptive control and Lyapunov based results which helps us to create stability and close-loop performance, which will not usually be guaranteed in an optimal framework, in the standard framework for offline identification of these parameters.

So, the other thing of course is that we do online identification and online tuning of the neural network. So, that is another thing that we are looking to do. So, we do this, if in a particular framework and this framework is what, we are going to look at. So, we already know what all these real numbers Rn and Rm cross n, and one important thing is that we, we are looking at functions f which go from some compact connected set in Rn to Rm and the set of continuous functions in this space S is denoted as Cm S.

Then we also, we have already seen the notion of infinity norm. So, this is essentially, the supreme norm is the same, it is similar, in fact. It takes this the, any norm of the vector norm of the function and it takes supremum of that over all the x in s, for all x in this domain, you take the supremum of the vector. So, this is what is called the supremum norm.

Then there is a Frobenius norm. We also saw this when we were looking in, at model reference adaptive control. So, the Frobenius norm of a matrix is just, the square of the Frobenius norm is the trace of A transpose A. It is not a vector norm, it is actually just a, it is essentially looking at the matrix as a vector and computing the, it somehow vector transpose itself of the matrix by stacking the matrix column, columns into a vector.

But of course, it is not a vector, it is not an induced norm, the way we have defined induced norms, I hope all of you remember. But it is in fact compatible with the induced norm. So, if you take the two norm of A x, then it is less than equal to the Frobenius norm of A times the two norm of x. So, this sort of inequalities are rather useful whenever we do sum of square kind of decompositions or some Cauchy-Schwartz type inequalities we want to use in a Lyapunov analysis, we know that these kind of inequalities are very useful for us.

Of course, we say that this matrix A of t is bounded if the infinity norm, induced norm of this matrix is bounded. Excellent. Now, we want to of course see what a typical neural network looks like. In this case, we are looking purely at the three-layer neural network. So, there are three layers, in the sense that there is an input layer here, there is an output layer here.

So, all these, sorry, let me be careful. So, there is a input here, there is a hidden layer here, and there is an output layer here. Input, hidden layer, output, whichever way you want to look at it. So, there is, so anything in the middle of input and output is called a hidden layer, it is as simple as that, because it is neither the input nor the output, so you do not look at it, so it is the hidden layer.

So, this is the three-layer neural network structure these. v i, v i js are the weights here, w i js are the weights here, and then you have these sigma's which are basically like activation functions. So, this is how neural network works. You have some n 1 inputs going in, then they are scaled by some weight here, and then they are summed up because you see there is a summing action here because of everything goes into everything else.

So, every input feeds into every, every node, if you may. So, so there is the scaling and then there is the summing action over these n 1 nodes, and then there is an offset which is added, and then it passes through this activation function. So, I think it is better to write it as this. So, the activation function acts on this, and once the activation function sigma acts on this, it again goes to the second layer.

So, of course there is again, weights, and that is the weights here. And then, then there is again a summation. Sorry, and then there is again an offset and then there is a summation. So, this is what is the output. So, so you can see that you have an input layer, a hidden layer and then an output layer. So, this is what is the three-layer neural network, one input, one output and a hidden layer in between.

So, there is one activation function sigma. So, so we have several things, you can see that are already unknown, what a typical neural network implementation would to identify are these offsets, and these weights because once I identify these offsets and these weights, of course, these are, these n 1s and n 2s are pre-chosen quantities.

So, once I identify these offsets and these weights, I have an exact, and almost exact relationship between the input and the output. So, so do not get confused by control systems technology. The input is not the control, output is not the output of the system and all that. This is for any non-linear function. For any non-linear function which is getting approximated in this. If you take any nonlinear function, it can be approximated in this. That is the whole idea, that s the whole idea.

Now, in this case we are doing the real-time neural network implementation, therefore we want them to exhibit learning while controlling. And we are not okay with learning first and then finding some parameters, good value the parameters then using it for control and so on as a separate problem, no. We are looking at jointly learning and control.

Now, what a standard choice of sig, these activation functions are, these like sigmoidal functions, these are like, sigmoidal functions are like smooth signum functions. Hyperbolic tangent function, we already know hyperbolic tangent functions, we saw them for the purpose of projection based adaptive controllers. And then you have these radial basis functions. These are essentially like the functions that you see in this bell curve in normal distributions and things like that.

So, all these are very nice functions for function approximation. That is why these choices are made. It is not just arbitrary functions. I would not take a linear function here for sigma. So, all these functions are somehow functions that help you to do good function approximation. So, so how we want to write this whole equation 1 is in a very compact, again, parameter regressor form. So, this looks like a, I would say, non-linear parameter regressor form. This looks like a non-linear parameter regressor. Why? So, how we do this is first of all, we define our x with an x 0. So, there are, you see there are n 1 xs here, but I add one more x to it as x 0.

And then what I do is, this sort of lets you to accommodate these thetas in this V. So, the first column of V contains a threshold theta w i. I am sorry, I was here. So, so if you have x 0 chosen as 1, vector of 1 then you can include this threshold vector into the V’s. So, if you look at, then if you compute V transpose x, it should be evident to you that because the first column of x is 1, then, sorry because x 0 is 1, not first column, because x 0 is 1, if I choose the first column of V as the theta Vs, then this is exactly what I have inside. And similarly, I can do for the w's also.

I take the sigma activation function, but again, the first term of this activation function is chosen, is just written as 1. So, the activation function, of course, if I write it as sigma z, then for a vector, it operates, it is assumed to operate element wise sigma z 1, sigma z 2, but if I make the first element to be 1, here, then, I can incorporate in w, this offsets also.

And once I do that, I can actually write y equal to w transpose sigma V transpose x, which is essentially like a non-linear parameter regressor form. So, nonlinear, because both w and V are parameters. It is not like a y theta equal to u kind of a thing, not as simple as that, but a little bit more complicated. So, so any tuning function basically w and V includes all the thresholds and the offsets and everything. So, that is the idea.

So, what did we see in this session? We sort of finished our discussion on this very interesting article which chronologically discusses how adaptive control developed, how it got connected with learning, how reinforcement learning and deep neural networks essentially are parameter, doing parameter identification in a different framework.

And now we started to look at the basic problem of doing real-time neural network-based control of a robotic manipulator. And here the neural network weights are going to be adjusted using our Lyapunov base adaptive control theory. And that is what, we are sort of trying to set up the framework for that. And in the subsequent session, of course, we will continue to look at this. So, I hope to see you again soon. Thank you.