INTELLIGENT CONTROL OF ROBOTIC SYSTEMS

Prof. M. Felix Orlando

Department of Electrical Engineering

Indian Institute of Technology Roorkee

Lecture 14: Robust RBFN

Good morning, everyone. Today, we are going to see robust RBFN. That is, robust radial basis function network. So, what is a robust radial basis function network? An RBFN is a type of artificial neural network that uses radial basis functions as its activation functions, which are present in its hidden layer.

It has only one hidden layer, and hence, it is a simplified neural network that can be useful for approximating non-linear functions. In general, RBFN consists of an input layer, a single hidden layer where the radial basis functions are applied, and an output layer that provides the final prediction or the classification that is required. Now, coming to the structure, as I mentioned, it has the input layer, hidden layer, and the output layer. The Gaussian function is the most commonly used RBF radial basis function, which is defined as $\varphi(x) = \exp\left(-\frac{\|x-c\|^2}{2\sigma^2}\right)$

where x is the given input vector, c is the center vector associated with each of the hidden layers or hidden neurons, and sigma is the spread of the Gaussian function. Other RBF radial basis functions are multi quadratic, which is given by $: \varphi(x) = \sqrt{\|x-c\|^2 + \beta^2}$

and inverse multi quadratic, which is given by $\varphi(x) = \frac{1}{\sqrt{\|x-c\|^2+\beta^2}}$

where beta in these multiquadratic and inverse multiquadratic functions is nothing but a parameter that affects the shape of the function. Such a network can be represented as follows, which is given by $f(X) = \sum_{i=1}^{r} w_i \varphi_i(\|X-\mu_i\|, \theta_i)$

where x belongs to n-dimensional Euclidean space is the input vector, phi i is the basis function of the network and wi's are the weight vectors connecting the hidden layer to the output layer, and mu i is called the center vector of the ith node. And theta i equal to theta i_1, theta i_2 up to theta i_n transpose is called the bandwidth vector of the ith node.

And here is the architecture where we have only one hidden layer, an input layer, and an output layer. Hence, it is a simplified one and can be used as a good function approximator. Now, coming to the training of RBFN. So, training can be done simply by

the pseudo-inverse technique. Or it can be done by the backpropagation method. The first step in training is to determine the centers. Centers can be determined using any clustering algorithm such as k-means. These centers are crucial as they define where the radial basis functions are centered. Then, compute the spread. The spread of the radial basis function, sigma, can be determined based on the distance between the centers.

A common approach to find sigma is the average distance between the centers. Now, the final step 3. Step 3 is to optimize the weights. The weights connecting the hidden layers to the output layer are optimized using methods such as linear regression or the gradient descent method. The goal is to minimize the error between the predicted output of the network and the actual desired output given to the network.

Now, coming to the major topic of function approximation. The process of estimating a function that best fits a set of data points is called function approximation. It is used in various applications like prediction, interpolation, and data smoothing. There are two main methods to do function approximation: one is parametric, and another is non-parametric. Now, let us see the first method, which is called the parametric method. The parametric method assumes that the relationship between inputs and output can be described by a predetermined mathematical model. Thus, the task is to find the optimal parameters for this model.

An example is linear regression, which models the relationship between the input and the output as a straight line. We assume that the function is of the form $y = \beta_0 + \beta_1 x$ and we have to find the parameters beta naught and beta 1 associated with this function. Advantages of this parametric method include being simple to implement and interpret, and computationally efficient, whereas the disadvantages are assuming a specific form of the function and having limited flexibility. Now, coming to the non-parametric method, these methods do not start with a predefined formula. Instead, they learn the relationship directly from the data, which is advantageous for handling real-world data.

That data exhibits truly non-linear relationships and also exhibits complex relationships. An example of this method is K-1. Nearest neighbors predict the value of a new data point based on the values of the k closest points from the training set. Advantages of this non-parametric approach include being highly flexible and capable of capturing intricate relationships within the data, and it does not rely on a specific functional form as in the case of the parametric method. The disadvantages are that it is computationally expensive

and requires a substantial amount of data in order to achieve better performance. Now, let us focus mainly on functional approximation using RBFN.

The radial basis function neural networks are a good candidate for function approximation because of their faster learning capability, as they have a simpler architecture compared to multi-layer neural networks. For function approximation, traditionally we use RBFN with Gaussian functions as the transfer functions and the method of least squares. This methodology works most of the time, but there are certain bottlenecks associated with using Gaussian functions directly as the activation functions. That includes difficulty in

The methods involving constant functions and the impact of outliers. These are the two bottlenecks associated with RBFN using Gaussian function as its activation function. First, let us see what is the difficulty associated with the constant functions. When the target function remains nearly constant in certain regions, a Gaussian function struggles to approximate that constant curve effectively without using a very large bandwidth. And this can lead to inefficiency of the network performance.

And also there is an impact of outliers. When the training data includes outliers, which means the value significantly different from the majority, the network's response in those regions is often inadequate due to the limitations of the least squares method. And you can see the results that have been taken from the work of Qin Cheng Li and In the year 1999, they proposed this RBFN-based approach for function approximation specifically because we are going to see RBFN for function approximation in the first phase of this lecture, and in the second phase, we are going to see how it can be utilized to control a single-link manipulator. So, here we can see the first figure.

It is basically the result of approximating a piecewise constant function. And the solid line is the desired one, and the dotted line is basically the network output. Similarly, figure B shows the poor approximation that is between the solid line and the network output, and you can see that in the second case we have outliers significantly visible, and it shows the performance is not very good in this function approximation. So, the solution here When we want to use the RBFN in order to have the functions that are constant in some regions and also having outliers, we may go for robust RBFN to tackle these two bottlenecks.

We can make the RBFN. By involving, instead of a Gaussian activation function, a composite of a set of sigmoidal functions can be used, which aims at approximating a

given function with nearly constant values. Use a method from robust statistics to reduce the impact of outliers. Then, the second one first one, in order to address the constant function. Layers of the functions, we may use a combination of sigmoidal functions.

The second one is, in order to address the effect of outliers, we can go for a method called Hample's m-estimator, which is efficient in handling large errors. Now, coming to the construction of a new activation function in one dimension, we are now familiar with the sigmoidal function, which is given by $f(x) = \frac{1}{1+e^{-bx}}$ where b is greater than 0,

that is the common sigmoidal function. And for obtaining a radial basis function, we combine two sigmoidal functions as follows, given by $g(x) = \frac{1}{1+e^{-\beta[(x-\mu)+\theta]}} - \frac{1}{1+e^{-\beta[(x-\mu)-\theta]}}$

The image of the next slide shows three different RBFs, radial basis functions, with the following values of the parameters, which involve mu equal to 3, beta equal to 5, 1.2, and 0.6, and theta is again a vector of 2.5, 1.5, and 0.5, respectively. For that, you have this profile. For g(x) with respect to the mu value. Okay.

So, we can see that g(x) for a given x can be obtained from these 3 rbfn sigma del functions. So, we have this performance given here. See here we can see that when either beta or theta are large, the function looks quite rectangular, which indicates that it is good for approximating constant functions. At the same time, the curves exhibit the properties of radial basis functions. Properties of radial basis functions so that they are valid activation functions for radial basis function neural networks.

So in higher dimensions, the proposed RBFN activation function can be defined as

$$g(X) = \prod_{j=1}^{n} \frac{1}{1+e^{-\beta_j[(x_j-\mu_j)+\theta_j]}} - \frac{1}{1+e^{-\beta_j[(x_j-\mu_j)-\theta_j]}}$$

Now, let us talk about the robust objective function. The method of least squares is not ideal when there are some outliers in the training pattern. The reason is as follows. Assume that theta is the parameter of the network which is being adjusted each time by minimizing a given function, say $E(r_p)$.

What is r_p? Residual. We will be seeing that in the upcoming slides. So, the adjustable parameter theta is given for each iteration by $\theta_{k+1} = \theta_k - \eta \sum_{p=1}^{P} \frac{\partial E(r_p)}{\partial \theta_k}$

Here, eta is a step size parameter.

You can also say the learning parameter and $E(r_p)$ is the objective function, and r_p is the expression that is given by the residual error that is given by

$$r_p = t_p - f(x_p)$$

Generally, the objective function requires an even symmetric property, that is, $E(0) = 0$ and continuity. The differential term in the above equation can be obtained as follows, which is given by $\sum_{p=1}^{P} \frac{\partial E(r_p)}{\partial \theta_k} = \sum_{p=1}^{P} \frac{\partial E(r_p)}{\partial r_p} \frac{\partial r_p}{\partial \theta_k} = \sum_{p=1}^{P} \psi(r_p) \frac{\partial f(X_p)}{\partial \theta_k}$ where $\psi(r_p) = \frac{\partial E(r_p)}{\partial r_p}$ and is called the influence function.

For acceptable performance, the difference between the output of a network and the desired output given to the network should be close to 0 or must be less than the tolerance value for all the training patterns.

But if there are no outliers, that means the errors that are very different from the rest of the data patterns. All the training data points will be accurately positioned, and their residuals, that is, the difference from the desired output, will be close to 0, which means $r_p \approx 0.$

But when outliers are present, their large residuals significantly affect the network performance because the network keeps adjusting its parameters to fit these outliers, which can distort the performance of the network. When outliers are present, the network struggles to accurately approximate the underlying function because minimizing the least squares criterion alone is not sufficient.

Now, to eliminate such problems, we use M estimators, which take the general form given by $\sum_{p=1}^{P} \frac{\partial \rho(r_p)}{\partial r_p} = \sum_{p=1}^{P} \psi(r_p) = 0$ In this case, we specifically use Hample's M estimator, whose influence function is given by

$$\psi(r) = \begin{cases} r, & a > |r| \\ a \operatorname{sgn}(r), & a \le |r| \le b \\ \frac{c - |r|}{c - |b|} a \cdot \operatorname{sgn}(r), & b \le |r| \le c \\ 0, & \text{otherwise} \end{cases}$$

Now, the following figure shows Hample's M estimator with A

Value given by 1, b being 3, and c being 4. This is the Hample's estimator, m-estimator for the given values of a, b, and c. Now, by observing the previous function, we can define a class of robust objective functions for the sigmoidal RBFN. Of the given form

$$E_R(r_p) = \sum_{p=1}^{P} [\varphi(r_p) - \varphi(0)]$$

where $\varphi(r_p)$ is a continuous function, $\varphi(0)$ is a constant, and p is the total number of input patterns. Now, the $\psi(r) = d\phi(r)/dr$ influence function will take the form

$$\psi(r) = s(r)t(r)$$

and S(r) is an odd symmetric monotonic continuous function, and T(r) is an even symmetric and continuous function that satisfies T(r) has a unique maximum at r equals 0, and $0 < t(r) \leq M$ where M is a real number, and T(r) increases strictly

$$-\infty < r \leq 0$$

And then $\lim_{r \to \pm\infty} s(r)/g(r) = 0$, where $g(x) = 1/t(x)$.

Here, to make this objective function adjustable, we can add an adjustable parameter in either S(r) or T(r). For example, $s(r) = r$

And $t(r) = e^{r^2/2\sigma}$

Now, coming to the learning of the network parameters, that means the parameter update tools, we use a gradient descent method for updating the parameters. That is, $\frac{\partial E_R(r_P)}{\partial w_i} = \sum_{p=1}^{P} \frac{\partial \varphi(r_p)}{\partial r_p} \frac{\partial r_p}{\partial w_i}$

Now, including all the parameters in a vector w, we can write the learning rule as

$$W(t+1) = W(t) - \eta \frac{\partial E_R(r_p)}{\partial W}$$

Here, t is the time step. And eta is the learning rate that varies from 0 to 1.

Now, coming to the adaptive growing technique of the network, for a radial basis function network to perform effectively, determining the appropriate size is required. However, it is difficult to achieve the optimal size because we do not know it. So, in that case, we need to go for an adaptive approach. This approach dynamically modifies the number of nodes according to specific guidelines. Adding nodes and deleting nodes are the methods that we will follow as part of the adaptability. So, add a node.

If the network is underperforming or learning too slowly, a new node is added at the point where the error of the residual is highest. This strategy allows the network to improve by concentrating on the areas with the most significant issues. Similarly, delete a node. After several tries or iterations, we remove nodes that have very little significance. These nodes do not contribute much to the learning process.

So, removing them will not affect the network's efficiency. Now, regarding algorithm implementation, the following are the definitions of the terms associated with the algorithm. Check period T, which means this tells us how often to check the network. If the network's number of iterations is a multiple of T, we check how the network is performing. Next, the objective function $E_R(i)$ this value shows how well the network

is learning during the iteration number I and the threshold delta entropy upper, which means delta, the threshold related to the upper limit. If the difference between the current and the previous values of the objective function is higher than this threshold, it means that the network is getting better. We then reduce the range of errors. Similarly, we have another threshold towards the lower value, which is delta entropy lower. If the difference between the current and the previous values of the objective function is smaller than this threshold, it means the network does not have enough nodes to learn well. So, in that case, we need to add a new node to the network.

Again, threshold weight, delta weight. Nodes with weight smaller than this threshold will be deleted. And finally, we have a threshold delta stop, which means to terminate the training process. With all these required determinations or definitions, we can define the algorithm as shown below. Step 1.

Set up the network initial conditions. Select and select the initial number of nodes for the network, set initial parameters associated with each node, and set ES equal to epsilon to a small value, where ES is used to record the value of the objective function, and set I equal to 1 and initialize the check period T. Then, step 2, construct the robust objective function, which is very essential here. Select proper s(r)and t(r). Compute extrema of the equation, which is $y(r) = s(r)t(r)$, by solving the equation $y'(r) = 0$

The extrema positions represent two cutoff points. Let these be plus or minus S. Let minus S to plus S be the initial confidence level of the residual. Then compute the objective function, which is the robust objective function ER of RP, by integrating Y of R. Then, step 3 involves computing the output by involving all the training patterns and finally computing the robust objective function by evaluating the value of the robust objective function of the network and storing it. Then, step 5: if I is a multiple of T (capital T), adjust the size of the network and the confidence interval of the residuals based on the following procedure.

If the norm or the magnitude of the discrepancy $|E_R(i) - E_S| > \delta_{\text{en\_upper}}$ between or the amount that we mention as a threshold value delta, then the objective function changes rapidly, reducing the confidence interval by finding new cutoff points, which is given by $\sigma_{\text{new}} = \begin{cases} \sigma_{\text{old}} \cdot r_d, & \text{if } \sigma > \sigma_L \\ \sigma_{\text{old}}, & \text{otherwise} \end{cases}$

where r_d is the decreasing rate and sigma L is the lower bound of the confidence interval of residuals. If the norm between the $|E_R(i) - E_S| > \delta_{\text{en\_upper}}$

the objective function does not change rapidly. Then, add a new node to the network by the adaptive growing technique with a memory cube.

Then, check the outgoing weight of each node and remove those nodes with outgoing weights smaller than the prescribed threshold. Because they do not have any significance in the performance of the training of the network. Then, store the current objective function ER of I to ES. Step 6: if $E_R(i) < \sigma_{\text{stop}}$,

that is the stopping threshold, then terminate the learning procedure. Otherwise, repeat by continuing to step 7.

Step 7 is to update the parameters of the network and continue with the iteration from step 3 onwards. Now, this is one of the simulation results that shows how the function approximation has been done with the M estimation. That is the original data, and the RBFN approximation is given by the red profile. Now, coming to the second part of today's lecture, which is robust control of a single-link manipulator using a radial basis function network.

Now, we know that the dynamic model of a general n-degree-of-freedom robotic system is given by $\boldsymbol{M(y)\ddot{y} + C(y,\dot{y})\dot{y} + G(y) = \tau}$

which means that m of y is the inertia matrix of size n cross n, which is a function of y, and the Coriolis matrix. Which is c of y comma y dot, which is a function of both y and y dot, is called the Coriolis and centrifugal matrix, and g of y is the gravity vector of size n cross 1, which depends on y, and tau is the joint torque vector of size n cross 1. In many practical purposes, m of y c of y comma y dot, and g of y are the network parameters which are unknown to us. In order to eliminate this problem, we use a radial basis function network to model these network parameter matrices m of y, c of y comma y dot, and g of y. If m of y, c of y comma y dot, and g of y are the ideal values of the respective matrices that is, the actual values of these matrices, and $\boldsymbol{M(y)}$, $\boldsymbol{C(y,\dot{y})}$ and $\boldsymbol{G(y)}$

are the outputs of the ideal radial basis function networks, then we can write the following equation, which is the ideal values $\boldsymbol{M(y)}$, $\boldsymbol{C(y,\dot{y})}$ and $\boldsymbol{G(y)}$

And the ideal radial basis function outputs $\boldsymbol{M(y)}$, $\boldsymbol{C(y,\dot{y})}$ and $\boldsymbol{G(y)}$ by this relationship,

which is

$$\boldsymbol{M(y) = M_{SNN}(y) + E_M}$$

$$\boldsymbol{C(y,\dot{y}) = C_{SNN}(y,\dot{y}) + E_C}$$

$$\boldsymbol{G(y) = G_{SNN}(y) + E_G}$$

So, Em, Eg, and Ec are the modeling errors of m of y, c of y comma y dot, and g of y. So, these are the modeling errors associated with m of y and M SSN of y. That means, the ideal values associated with the model as well as the ideal network output values. That means the discrepancy between the ideal values of the model and the ideal values of the ideal radial basis function network output. Now, we can write the output of the ideal radial basis function network as follows

$$\boldsymbol{M_{SNN}(y) = \{W_M\}^{\mathrm{T}} \cdot \{\xi_M(y)\}}$$

$$\boldsymbol{C_{SNN}(y,\dot{y}) = \{W_C\}^{\mathrm{T}} \cdot \{\xi_C(y,\dot{y})\}}$$

$$\boldsymbol{G_{SNN}(y) = \{W_G\}^{\mathrm{T}} \cdot \{\xi_G(y)\}}$$

On substituting the above equations in the dynamic model, we get

$$\boldsymbol{M(y)\ddot{y}_r + C(y,\dot{y})\dot{y}_r + G(y) = M_{SNN}(y)\ddot{y}_r + C_{SNN}(y,\dot{y})\dot{y}_r + G_{SNN}(y) + \mathbf{E}}$$

$$\boldsymbol{= [\{W_M\}^{\mathrm{T}} \cdot \{\xi_M(y)\}]\ddot{y}_r + [\{W_C\}^{\mathrm{T}} \cdot \{\xi_C(y,\dot{y})\}]\dot{y}_r + [\{W_G\}^{\mathrm{T}} \cdot \{\xi_G(y)\}] + \mathbf{E}}$$

That weight matrix associated with the gravity term of the manipulated dynamic model associated with a neural network and the output psi g of y, that is the hidden layer output of the neural network associated with the gravity term plus E. Here $\boldsymbol{E = E_M\ddot{y}_r + E_C\dot{y}_r + E_G}$

Obviously, the radial basis function network is in practice not ideal, and hence we can only estimate the values of m s n n of y, c s n n of y comma y dot, and g s n n of y. Instead of these ideal values, we are going to estimate them. So, the estimation can be represented by a hat function.

$$\boldsymbol{\widehat{M}_{SNN}(y) = \{\widehat{W}_M\}^{\mathrm{T}} \cdot \{\xi_M(y)\}}$$

$$\boldsymbol{\widehat{C}_{SNN}(y,\dot{y}) = \{\widehat{W}_C\}^{\mathrm{T}} \cdot \{\xi_C(y,\dot{y})\}}$$

$$\boldsymbol{\widehat{G}_{SNN}(y) = \{\widehat{W}_G\}^{\mathrm{T}} \cdot \{\xi_G(y)\}}$$

Here, W_M hat, W_C hat, and W_G hat are the estimates of the ideal values WM, WC, and WG, respectively. Now, the controller design here is first, let us define the error terms $e(t) = y_d(t) - y(t)$

$$\dot{y}_r = r(t) + \dot{y}(t)$$

$$\ddot{y}_r = \dot{r}(t) + \ddot{y}(t)$$

Now, defining R, that is a filtered error given in terms of the tracking error, that is given by $r = \dot{e} + \Lambda e$ we have $\dot{y}_r = \dot{y}_d + \Lambda e$

$$\ddot{y}_r = \ddot{y}_d + \Lambda \dot{e}$$

Now, we propose the following controller that is given by $\tau = \tau_m + K_p r + K_i \int r\, dt + \tau_r$

Here, tau_m is the model estimated control law that is given by $\tau_m = M_{SNN}(y)\ddot{y}_r + C_{SNN}(y, \dot{y})\dot{y}_r + G_{SNN}(y)$

And tau_R is the robust term in the control law that is designed as $\tau_r = K_r \mathrm{sgn}(r)$

Now, to find the adaptive law for updating the weights of the networks, we have to perform the Lyapunov stability analysis of the control law proposed, which leads to the following update law. That is

$$\dot{\hat{W}}_{Mk} = \Gamma_{Mk} \cdot \{\xi_{Mk}(y)\}\ddot{y}_r r_k$$

$$\dot{\hat{W}}_{Ck} = \Gamma_{Ck} \cdot \{\xi_{Ck}(y, \dot{y})\}\dot{y}_r r_k$$

$$\dot{\hat{W}}_{Gk} = \Gamma_{Gk} \cdot \{\xi_{Gk}(y)\} r_k$$

That is gravity term, Coriolis term, and the inertia terms. So, we have these three update laws associated here. Here, gamma_mk, gamma_ck, and gamma_gk are the learning rates associated for the respective RBF networks. Now, with this control design, we have performed a simulation of a simple single-link manipulator whose dynamic model is given by $M(y)\ddot{y} + C(y, \dot{y}) + G(y) = \tau$

Let m be 0.1 plus 0.06 times sin of y and c be 3 times y dot plus 3 cos of y and g being m g l cos of y with m equal to 0.02, l equal to 0.05, and g equal to 9.8. Here, the initial states are y of 0 equal to 0.15 y dot of 0 equal to 0, and let the desired trajectory be a sine trajectory. Here, we propose a network, an RBF network, with two inputs, y and y dot, and there are seven hidden neurons and one output. With the input being y and y dot, as I mentioned, the parameters of the Gaussian functions c i and b i are designed as minus 1.5, minus 1.0, minus 0.50 up to 1.5 and 20, respectively. These are the centers, and this is the bias value. The initial weight

values are chosen to be 0. It can be chosen sometimes as random values as well, but in this simulation, we have chosen 0. We use the control law that is proposed and the adaptive law with the following parameters: kr being 5 into 10 power minus 4, kp being 0.1, ki being 1 into 10 power minus 3, and lambda being 0.8. The learning rates are gamma m is 5 into 10 power minus 3, again gamma c being the same value with gamma g being 5 into 10 to the power minus 3. With this, we could get the desired trajectory tracked by the single-link manipulator to track the desired sine wave, that is, the joint angle from the network Q, and the desired joint angle is given by sine of t, which is a position trajectory. Also, we checked the velocity trajectory, which is coming out to be tracking successfully. And then, the control input to track the decided sine trajectory is also given in the form of a sine, which is varying with respect to time.

And then, the estimated values of the ideal values of the network parameters, which are nothing but the dynamic model parameters, which correlate to the centrifugal matrix C and the gravity vector, are shown here with the blue line being the ideal value and M or the red line being the estimated values from the network, okay. And here, we could observe a severe discrepancy between the ideal network values and the estimated network values associated with the dynamic model of the manipulator, and this is because the desired trajectory lacks persistent excitation, which is a common bottleneck in practical applications. With this, I conclude today's lecture that we have seen robust radial basis function towards

Function approximation and control of a single-link robot manipulator. With this, I conclude today's lecture on robust radial basis function neural networks. Thank you so much for your time.