INTELLIGENT CONTROL OF ROBOTIC SYSTEMS

Prof. M. Felix Orlando

Department of Electrical Engineering

Indian Institute of Technology Roorkee

Lecture 11: Neural Network based Robot Control

Good morning, everyone. Today, we are going to see a lecture on Neural Network-Based Robot Control. The outline of this lecture will be as follows. First, we have the introduction to neural networks and the biological inspiration behind ANN, artificial neural networks. Then, we are going to see perceptrons, followed by multilayer perceptrons.

Then, we will see multilayered neural networks, their architecture, and then we will discuss the principle of gradient descent. Next, we move on to the backpropagation algorithm, which is an important algorithm for neural network-based robot control. Then, we will see inverse kinematic control using neural networks with an experimental demo. And finally, we will conclude our lecture today. Thank you.

So, we will start with the introduction part. First, we start with biological neural networks. The important thing is nerve cells. There are two types of nerve cells: glial cells and neurons. The glial cells do not take part in transmitting or receiving messages; rather, they protect the neurons, and each yellow structure represents a neuron here in this schematic.

The neuron is a functional unit of the nervous system. There are approximately 100 billion neurons in the human brain. Now, coming to the neuron structure, the pyramidal structure cell, which is a common neuron, is given here. Each neuron has three parts: dendrites, cell body, and axon. Dendrites receive information from another cell and pass it to the cell body. This message, which it has received, is processed in the cell body. The cell body contains the nucleus, mitochondria, and other organelles typical of eukaryotic cells. The axon conducts messages away from the cell body. Let us talk about the synapse now.

The junction between a nerve cell and another cell is called a synapse. The messages that travel within this neuron are basically electrical actions or action potentials. Messages travel within neurons through action potentials, which are electrical signals. The space

between the two cells is known as the synaptic cleft. In order to cross the synaptic cleft, it is required to have the actions of neurotransmitters, which are stored in small synaptic vesicles clustered at the tip of the axon.

Now, let us see the basic functions of our biological nervous system. They receive sensory input from the external as well as internal environments, and they integrate the received input. Finally, they respond to the stimuli by producing a response or output. So, the biological inspiration is that living organisms react adaptively to the changes in their external as well as internal environment. And they use their nervous system to perform these behaviors.

Now, the artificial neural network must replicate this behavior. Thus, the artificial neural network is basically an approximate model or simulation of the nervous system that should be able to produce similar responses and behaviors in artificial systems. Let us talk about the fundamental unit of an artificial neural network, which is a neuron. A single neuron is called a perceptron. So, it has a structure like this.

It has a summer structure. The output of the summer goes to a function called the transfer function, and then we get an output. The summer receives two inputs. One is the given input, scalar input multiplied by the scalar weight, which will become one part of the input to the summer, and the other term of the summer is the input 1 multiplied by the bias. That is why 1 into b. So, the input that the summer receives is two terms.

Basically, the first term $wx+b$ and that input goes as input to the transfer function. The transfer function has this input N, which is $wx+b$ as its input argument. Accordingly, we get the output A based on the transfer function. Likewise, for the multi-input neuron, we have the neuron structure with

So, many inputs. So, many inputs are connected to the summer through a weight vector. So, the input to this summer is basically the W vector, the weight vector multiplied by the input vector plus B, which is the bias, the input multiplied by the bias. So, that acts as the input to the transfer function, which is F. Thus, a is $f(wx+b)$ where w is a vector called the weight vector, x is the vector called the input vector, and b is the bias input.

Now, let us see about the activation function or transfer function. In general, they take these forms. One is a linear transfer function, which means the output is the input, which is phi of z equal to the given input z, and the logistic activation function is a sigmoidal activation function, which takes the range 0 to 1, which has the mathematical expression

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Similarly, we have another form of the sigmoidal function, which is hyperbolic tangent function,

which is given by $\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ which takes this output range varying from minus 1 to plus 1. The profiles of the linear activation function, the logistic sigmoidal function, and the tan hyperbolic function are all given here. As I said, the linear activation function has a linear response. The logistic sigmoidal function will have a sigmoidal curve varying from 0 to 1. The tan hyperbolic activation function will have a curve varying from minus 1 to plus 1 as its response.

And because the linear function produces output based on the input, which is linear in relationship, it is limited in its application to complex neural networks. Whereas, the sigmoidal activation function is applicable in complex neural networks, where its range is between 0 and 1. Similarly, the hyperbolic tangent function, which is given by the expression $\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ has its response varying between -1 and +1. Now, coming to the fundamental application of artificial neural networks, they are used for classification and regression. Classification predicts categorical classes, precisely coming under the discrete domain.

The basket having different colored balls or apples will be segregated based on the colors of the apples. So, they are coming under three classes, let us say four classes. So, each class 1, 2, 3, 4 becomes a discrete quantity. And in regression, it models continuous-valued functions. Thus, classification is to identify which class, and regression is to estimate a continuous quantity.

Thus, classification is finding a function that maps input into classes. Thus, class equals function of input. Given an input x, it belongs to which class is the output of the classification. For example, if you take a hand exoskeleton to move the fingers towards grasping and releasing. So, grasping is one class, and releasing is another class towards the on-off controller.

Based on the input subject intention through the brain signal or through the muscle signal, the exoskeleton will help elderly human beings to grasp objects or to release them. That is the output of classification through neural network application. And regression is to find the function that maps input into values. Thus, the value is basically a function of the given input x. And here, the robotic example is inverse kinematics control. Given a

desired position to reach, you get the corresponding joint angle vector of the robotic system to reach that position in the Cartesian space.

And thus, we have a continuous mapping between the Cartesian space and the joint space. Next, let us see the architecture of the fundamental neural network. Let us take an L-layered neural network. You can see the input layer where the inputs are X_1, X_2, X_3 up to X_m, which means m-layered inputs. And we have the hidden layers numbering n minus 1 hidden layers and 1 output layer.

The output layer is n. Thus, the hidden layers are l_1, l_2, l_3, and it goes on. Whereas the output layer is the last layer, which is l suffix capital n. Thus, it is an L-layered neural architecture. Neural architecture. So, we can see here the L-layered architecture consists of an input layer of source nodes, L minus 1 hidden layers, and an output layer which is the Lth layer. If it is a three-layered network, the third layer is the output layer, and two hidden layers are there, and one input layer.

The synopsis of these notations used to represent a multilayer neural network with L layers is given as, let X be the m cross 1 input vector and Y be the output. We can say, instead of Y, we can say it is O, O being the output, small o. And Lk, where k varies from 1 to L, is the index for representing a neuron in the kth layer. And W L K L K minus 1 represents the synaptic weights connecting the L Kth neuron of the Kth layer to the L K minus 1th neuron of the K minus 1th layer. Which means that W L 1 L naught implies the synaptic weights that connect the L 1 neurons to L naught neurons. Now, let us see the principle of gradient descent.

Here, we need to calculate the derivative or the gradient of the error with respect to the weights and then change the weights by a small increment in the opposite direction of the gradient. Thus, the rule here is the gradient:

$$W^{new} = W^{old} - \eta \frac{\partial E}{\partial W}$$

So, in this profile between the error function and the weight vector, we need to find the optimal weight vector where the error function is minimized. So, we have here the negative slope and the positive slope in this direction. So, when we are here

We have to increment delta W positively. When we are here in the W, let us say initial W is here, then we have to go towards this point for this optimal value and hence delta W here is decremented. So, for delta W, we take this expression minus eta into dou E by dou W, where dou E by dou W is a slope. So, here if you want to do delta W, it must be positive when we start from here to reach the optimal value where it corresponds to the

minimum error function or the cost function. So, here it must be positive whereas the slope is a negative slope.

So, delta W being minus eta dou E by dou W becomes minus eta into minus value because of the negative slope. So, it becomes a Thus, starting from here leads towards a positive increment of delta W. Similarly, when we start from this point, delta W is decremented towards the optimal value of W. So, for both cases, this rule obeys where eta is the learning rate. It varies from 0 to 1. Value for eta learning rate.

Higher the learning rate, faster is the convergence. Smaller the learning rate, convergence is slow. So, here the cost function is defined as $E = \sum_{p=1}^{p=N} E^p$

where E^p is given by $E^p = \frac{1}{2}(o_p{}^d - o_p)^2$

where O_P and $O_P{}^d$ denote the actual and desired patterns, and it has a learning rate as I told you that varies from 0 to 1. Next, we see the backpropagation algorithm.

So, the objective is to adjust the weights of the network so as to minimize the cost function, which will train the network for the mapping of the required function. For function approximation, we need to train the network, which means the adaptation of the network parameters towards the given input. So that the network can be able to approximate any given input function. So, through the backpropagation algorithm, we are going to see how the network parameters are updated.

And the parameters of the network getting updated lead to a term called training of the network or network learning. The backpropagation algorithm uses the principle of gradient descent to train the network parameters. Let us define the output, input, and weight parameters associated with the network. x being the m cross n input vector, o being the n cross 1 output vector, and w_i k i k minus 1 represents the weight connecting the i kth neuron of the kth layer and the i k minus 1th neuron of the k minus 1th layer, and the h_i response, which is nothing but the response of the ith neuron in layer 1. Let us see here.

Let us consider To derive this backpropagation algorithm, a three-layered network x1 to xm is an input layer where the sources are x_1 to x_m, which means x is an m cross 1 input vector, and we have three layers l_1, l_2, and l_3, where l_1 and l_2 are the hidden layers and l3 is the last layer called the output layer, so this network has one Input layer, 2 hidden layers, and 1 output layer. And you can see here the weights connecting the first hidden

layer L1 to the input layer L0 are represented by WL1L0. This is the nomenclature that I give to the weight vector connecting the first hidden layer to the input layer.

Likewise, for the second hidden layer to the first hidden layer, the weight vector is given by WL2L1. And WL3L2 is the weight vector connecting the output layer L3 to the second hidden layer L2. And you can see the outputs of the first hidden layer are given by H1, H2, H3 up to HN1, where there are N1 number of neurons in the first hidden layer. Likewise, the output of The second hidden layer is also given by H1, H2 up to HN2 because there are N2 numbers of hidden neurons in layer 2.

So, what is observed here is the neuron's output is given by H1, H2, and so on, which are precisely the hidden layer neuron's output. Whereas, The output of the neurons in layer 3, represented by H1, H2, H3, are basically the outputs of any neurons. Because the third layer is the output layer, H1 is represented as O1, H2 as O2, H3 as O3, and Hn3 as O5. Accordingly, we have the output of the neurons of the output layer represented by O2, O1, O3 up to ON3.

Next, we have two phases in training the neural network in Back propagation-based algorithm. The first one is the forward phase, which means we are getting the response or output of the neurons present in the hidden layer as well as the output layer based on the weight vectors as per the input given. Given an input, getting the response is called the forward phase. Here, the input to the L1th neuron of the first hidden layer is S_{L1}, which is given by

$$s_{L_1} = \sum_{L_0=1}^{n} W_{L_1 L_0} x_{L_0}$$

And the output of the L 1 th neuron of the first hidden layer is H_{L1}, which is given by the transfer function of the input S_{L1}. Since we use the transfer function being the sigmoidal transfer function which takes the form $h_{L_1} = \Psi(s_{L_1}) = \frac{1}{1 + e^{-s_{L_1}}}$ where the input is S_{L1}. That is why it is minus S_{L1}.

Likewise, the input to the L second neuron of the second hidden layer is S_{L2}, which is given by

$$s_{L_2} = \sum_{L_1=1}^{n_1} W_{L_2 L_1} h_{L_1}$$

And the output of the L2 neuron of the second hidden layer is h_{L2}, which is the transfer function of S_{L2}, which is the input. That is given by the sigmoidal transfer function

$$h_{L_2} = \Psi(s_{L_2}) = \frac{1}{1 + e^{-s_{L_2}}}$$

which is S_{L2}. Next, the forward phase continues. The input to the third layer, which is the output layer, is S_{L3}, which is given by $s_{L_3} = \sum_{L_2=1}^{n_2} W_{L_3 L_2} h_{L_2}$

and the output is given by the output layer O_{L3}, where L_3 is the output layer and O is the output layer from that network. So, OL3 equals the transfer function of S_{L3} input, thus

$$o_{L_3} = \Psi(s_{L_3}) = \frac{1}{1 + e^{-s_{L_3}}}$$

Now, let us see the backpropagation of error. This is the starting phase of the backward phase, forward phase, and the backward phase. The instantaneous cost function in error is given by $E(t) = \sum_{L_3=1}^{n_3} E_{L_3}(t) = \frac{1}{2} \sum_{L_3=1}^{n_3} (o_{L_3}{}^{d}(t) - o_{L_3}(t))^2$

So, the weight update starts here based on the backward propagation of the error. So, the update of the weights connecting the output to the hidden layer is the first step because, from the back side, the first layer that we are going to face is the second hidden layer and the output layer. So, the weights connecting the second hidden layer and the output layer are to be first updated. So, $W_{L_3 L_2}(t+1) = W_{L_3 L_2}(t) - \eta \frac{\partial E(t)}{\partial W_{L_3 L_2}(t)}$

where this term, the second term $\frac{\partial E(t)}{\partial W_{L_3 L_2}(t)} = \sum_{L_3=1}^{n_3} \frac{\partial E_{L_3}(t)}{\partial o_{L_3}} \times \frac{\partial o_{L_3}}{\partial W_{L_3 L_2}(t)} \cdots$

where the chain rule plays a vital role. So, for the sigmoidal activation function, the transfer function of the given input is given by 1 by 1 plus e power minus SLXLK, which is the input to the activation function. Thus, the partial derivative of this with respect to its input is given by an expression: $\Psi(s_{L_k}) = \frac{1}{1 + e^{-s_{L_k}}}$

Thus, it leads to $\Psi - \Psi^2$ which is nothing but the function into 1 minus the function.

Thus, $\frac{\partial \Psi(s_{L_k})}{\partial s_{L_k}} = (1-\Psi)\Psi$

So, from the cost function of the error, we can write $\frac{\partial E_{L_3}(t)}{\partial o_{L_3}} = -\frac{1}{2} \times 2(o_{L_3}{}^d(t) - o_{L_3}(t))$

since error is $E_{L_3}(t) = \frac{1}{2}(o_{L_3}{}^d(t) - o_{L_3}(t))^2$

That is why this has become your partial derivative of E_{L3} with respect to O_{L3}. Then, the second term $\frac{\partial o_{L_3}}{\partial W_{L_3L_2}} = \frac{\partial o_{L_3}}{\partial s_{L_3}} \times \frac{\partial s_{L_3}}{\partial W_{L_3L_2}} = o_{L_3}(1-o_{L_3}) \times h_{L_2}$

This is obtained because of the sigmoidal activation function's derivative with respect to its input. Then, combining equations 3 and 4, the previous equation 2 becomes

$$\frac{\partial E(t)}{\partial W_{L_3L_2}(t)} = -(o_{L_3}{}^d - o_{L_3})o_{L_3}(1-o_{L_3})h_{L_2}$$

So, this portion into this portion with a minus sign is what is combined here. Now, we can see that the weight update law $W_{L_3L_2}(t+1) = W_{L_3L_2}(t) + \eta(o_{L_3}{}^d - o_{L_3})o_{L_3}(1-o_{L_3})h_{L_2}$

So, it is this expression. So, this expression, equation 6, can be written as equation 7, where $W_{L_3L_2}(t+1) = W_{L_3L_2}(t) + \eta\delta_{L_3}h_{L_2}$

Likewise, we can see the update of the weights connecting between the two hidden layers, that is, layer L2 and layer L1. Thus, the weight update law becomes

$$W_{L_2L_1}(t+1) = W_{L_2L_1}(t) - \eta\frac{\partial E(t)}{\partial W_{L_2L_1}(t)}$$

where $\frac{\partial E(t)}{\partial W_{L_2L_1}(t)} = \sum_{L_3=1}^{n_3}\frac{\partial E_{L_3}(t)}{\partial o_{L_3}} \times \frac{\partial o_{L_3}}{\partial W_{L_2L_1}(t)}$ Let us take this expression first $\frac{\partial o_{L_3}}{\partial W_{L_2L_1}} = \frac{\partial o_{L_3}}{\partial h_{L_2}} \times \frac{\partial h_{L_2}}{\partial W_{L_2L_1}}$

So, this term is further expanded into $\frac{\partial o_{L_3}}{\partial h_{L_2}} = \frac{\partial o_{L_3}}{\partial s_{L_3}} \times \frac{\partial s_{L_3}}{\partial h_{L_2}} = o_{L_3}(1-o_{L_3}) \times W_{L_3L_2}$

whereas input to the third layer with respect to output from the second layer gives you W_{L3L2}.

So, we know that O_{L3} is given by this expression because of the sigmoidal activation function, and $o_{L_3} = \frac{1}{1+e^{-s_{L_3}}}$

S_{L3} is given by this expression $s_{L_3} = \sum_{L_2=1}^{n_2} W_{L_3L_2} h_{L_2}$

And thus, $\frac{\partial h_{L_2}}{\partial W_{L_2L_1}} = \frac{\partial h_{L_2}}{\partial s_{L_2}} \times \frac{\partial s_{L_2}}{\partial W_{L_2L_1}} = h_{L_2}(1 - h_{L_2}) \times h_{L_1}$

Now, combining equation 13, 12, and 3, equation 10 can be written as ∂E/∂WL2L1 becomes finally, $-h_{L_2}(1 - h_{L_2}) h_{L_1} \sum_{L_3=1}^{n_3} \delta_{L_3} W_{L_3L_2}$ which is the 14th equation.

Thus, the weight update law for the second weight connecting L2 and L1 layers becomes

$W_{L_2L_1}(t+1) = W_{L_2L_1}(t) + \eta \delta_{L_2} h_{L_1}$ where $\delta_{L_2} = h_{L_2}(1 - h_{L_2}) \sum_{L_3=1}^{n_3} \delta_{L_3} W_{L_3L_2}$

which is equation 16. Now, finally, we can go for the update of the weight vector connecting the first hidden layer to the input layer that is H_{L1}. L1 and L0. Thus,

$$W_{L_1L_0}(t+1) = W_{L_1L_0}(t) - \eta \frac{\partial E(t)}{\partial W_{L_1L_0}(t)}$$

In the same way, when we continue this, we end up the weight update rule for this weight vector which is connecting L1 and L0 becomes $W_{L_1L_0}(t+1) = W_{L_1L_0}(t) + \eta \delta_{L_1} x_{L_0}$

where $\delta_{L_1} = h_{L_1}(1 - h_{L_1}) \sum_{L_2=1}^{n_2} \delta_{L_2} W_{L_2L_1}$ which is equation 16.

Now, let us see the generalized delta rule can be obtained in terms of this example. Let us consider a four-layered network where delta L4 represents $\delta_{L_4} = (O_{L_4}{}^d - O_{L_4}) O_{L_4}(1 - O_{L_4})$

$$W_{L_4L_3}(t+1) = W_{L_4L_3}(t) + \eta \delta_{L_4} h_{L_3}$$

Likewise, we can go for the third layer update rule, which is $W_{L_3L_2}(t+1) = W_{L_3L_2}(t) + \eta \delta_{L_3} h_{L_2}$

Likewise, the second layer in terms of delta L_2, where delta L_2 is written in terms of delta L_3. $W_{L_2L_1}(t+1) = W_{L_2L_1}(t) + \eta \delta_{L_2} h_{L_1}$

Whereas, for the first layer, we can say which is connecting the weights between layer L_1 and L_0, the input layer and the first hidden layer.

Thus, it is given by $$W_{L_1 L_0}(t+1) = W_{L_1 L_0}(t) + \eta \delta_{L_1} x_{L_0}$$

And thus, $$\delta_{L_1} = h_{L_1}(1 - h_{L_1}) \sum_{L_2=1}^{n_2} \delta_{L_2} W_{L_2 L_1}$$

Now, let us come to the application of neural networks towards inverse kinematic control utilizing the backpropagation algorithm that we have seen before. So, here it is basically meant for the inverse kinematic control problem where the robotic system is here.

In the Cartesian space, we have X actual, Y actual. It has to go to the X desired, Y desired point in space. So, given this space, we need to Move this robotic system to reach here by providing the joint angular vector containing theta 1 and theta 2. This is now theta 1 updated and theta 2 updated.

So, to solve this inner schematic control problem, we use this multi-layer neural network with BPN. Given the desired position in space x_d, y_d, and z_d in 3D Cartesian space, the network weights are updated, which connect between the output layer and the hidden layer, and the hidden layer and the input layer, and so on. So, given the input with the associated weights, the network is updated. With the network parameters, we get an output. That output of the network is considered as the joint angular vector of the robotic system, and that joint angular vector is given as input to the forward kinematic model of the robotic system. Hence, we get an actual value of the robotic tip for the desired tip position. The error between the desired and the actual tip position obtained from the forward kinematics through the input by the network output is the error function that has been backpropagated to update the weight parameters connecting the hidden to the output layer and the input to the hidden layer.

These two weights are updated. Then, with the updated weight parameters, another input vector representing a different Cartesian position for the robotic system is given. Then, the process is continued until the error is minimized under a tolerable value. That is how the training of this network happens. This procedure is almost a semi-supervised or semi-unsupervised learning.

This procedure is used to get the results. With this, we can track the given desired trajectory precisely. Here is the pseudocode for this inverse kinematic control based on neural networks. Given the desired value, continue the loop with a condition for stopping

the loop: if the error is less than a tolerant value, stop the loop; otherwise, continue with the loop.

The network output for the given input is basically the joint angle that has been provided to the forward kinematic model in order to obtain the actual network output because we need to compare the desired Cartesian position with the actual Cartesian position of the robotic system. And the error is backpropagated, which is given by this expression, in order to update the network parameters. With the updated network parameters, we provide another new desired Cartesian position for the robotic system to reach. This process continues until the error is less than a tolerable value.

Now, this is the demo that we have performed using this type of semi-unsupervised learning technique to solve a regression problem for the inverse kinematic control of a 5-degree-of-freedom follower robot in minimally invasive surgery. The demo is presented here, where we have controlled the follower robotic system by providing a master command through the haptic device. The follower robot, which has 5 degrees of freedom and 5 links, must follow the given tip trajectory. This trajectory has been tracked by providing the joint angular values obtained from the neural network, and we achieved this through this simple neural network approach. Coming to the conclusion of this lecture, we first introduced neural networks, then discussed the backpropagation neural network algorithm, followed by the generalized delta rule, and finally applied

neural networks to the inverse kinematic control of a robot. Thank you very much.