Optimal Control of HRI system using RL and Neural Network

Shubham Miglani Ahmad Alhaji

I. MOTIVATION

The base paper deals with an intelligent human-robot interaction system. The problem of finding the optimal parameters of the model is transformed into an LQR problem which minimizes the human effort and optimizes the closed-loop behavior. As the human model is difficult to estimate, Reinforcement learning is used in the paper to solve the LQR problem. The plan for the project consisted of two tasks. The first part was the application of integral reinforcement learning and the second part was using a neural network to find the optimal solution for LQR.

II. Reinforcement Learning

As per our initial understanding, the integral reinforcement learning method was different from the actor-critic method taught in class. But as we explored Frank Lewis papers and reading about integral reinforcement learning in the book, we found that they are the same only.

Firstly, we used the code provided on moodle with our system which is a 6 by 6. We transformed the weight equations in the code. Q and R matrices were taken as identity as they were not provided in the base paper. Initial conditions were also assumed. After simulating the code, we faced problems that even though the states go to zero, controller gain does not converge to the optimal one. And if the states remain at zero, the difference between two intervals of states becomes very less. As this difference term is in the division, it increases the controller gain significantly leading to incorrect behavior of the system states. As per our understanding, sampling time greater than the required number of samples should provide enough information for the controller gain to converge but it wasn't the case.

We tried the same code with a 4by4 car engine system and a 5by5 aircraft system. We were able to make the code work for the car engine system but only after making modifications in the sampling time and number of samples. Also, the initial K was having an impact on the convergence to an optimal one.

Due to all the above problems, we decided to use the reinforcement library provided in Matlab. It is based on the actor-critic method only. Our system was converted to a discrete one based on the first-order hold. After creating a discrete environment of the system, a custom LQR agent is defined using Q, R and initial stabilizing K. The simulation results using this method were very consistent. The optimal K was found to converge to the optimal one obtained from the dlqr command (for discrete systems). The same method was tried for various systems as mentioned above and found to be working for all of them.

III. Neural Network: Approach 1

The next task was to use a neural network to find the optimal controller gain or the solution of the LQR problem. As per our discussion, our first approach was to use A and B matrices as input to the neural network and K matrix as the output. So, the first part was Data generation for training.

1) Data Generation

For Data Generation, we used the rss command in Matlab. It gives random state-space system models that are stable. We had to add logic to ensure that the system is also controllable, otherwise, the LQR command would give an error. We wrote a general-purpose code that generates the data for any input matrix size and any number of data points.

2) Model Training

Due to our prior experience in dealing with neural networks in python, firstly we started with that. The input to the neural network were 48 values (A(6by6) and B(6by2)). The output was 12 values (K(2by6)). For the initial training 100 data points were taken. Adam optimizer was used with 32 neurons in the hidden layer with a learning rate of 0.001. The loss function used was the Mean Squared error.

3) Results

After training, it was found that the loss would not go below 0.01. This was a little confusing as the data is linear, the neural network should ideally be able to learn a linear function perfectly even if it overfits. So, we increased our data points to 1000 and found the loss decreased to 0.006. Considering 10 times the initial data points, the decrease in loss was quite insignificant. Also, we made the data samples 10000, then the loss further decreased to 0.005. After exploring with learning rate, the number of hidden layers, number of neurons, etc. we couldn't find a way to make the loss converge to a very small value. We tried the same dataset in Matlab and found that it gave the same results. The Matlab Neural net fitting toolbox was used which just takes data. There is no coding required in that, but it also gave us the same results that we achieved with python. So, we could conclude that the problem was not in the code but maybe with our data. So, we went to the second approach.

IV. Neural Network: Approach 2

The next approach was to use states as input to the neural network and optimal control as the output. So, again the first part was Data generation for training.

1) Data Generation

For Data Generation, building on the first approach, we used the random state-space models and used to simulate the states and further obtain the optimal control input by using the gain from lqr command. The data was taken for 100-time samples.

2) Model Training

The input to the neural network were 600 values (states(6by100)). The output were 200 values (input(2by100)). A similar approach and parameters were used as in the first approach.

3) Results

This time we started firstly in Matlab as training is much easier in that. But we found that the training was very slow possibly due to many input-output data (6*100*10000). We were aware of using GPUs for running neural networks on google colab environment with python, so we used them instead and the training was much faster on it.

We observed the same problem as last time with the loss. The loss would reach a minimum value and would not go below it.

As per our understanding, it was possibly because of the method we are using to generate data. Maybe there are redundant inputs or the way Matlab randomly initializes them doesn't let the network to converge to the optimal solution. So we moved to approach 3.

V. Neural Network: Approach 3

In this approach, our thinking was to use a system whose values depend on certain parameters such as in inverted pendulum, the mass of the cart, pendulum and many other parameters are there. We decided to use 2 parameters as variables and generated different A and B matrices by changing these parameters and used them for training. The parameters were chosen in a range let's say between -100 to 100 and data generated between those points.

1) Training

We trained the neural network in Matlab with 2 hidden layers having 12,20 neurons respectively. The input to the neural network was 2 values (the parameters being varied) and the output was the optimal K. Firstly we simulated this for our base paper. There were human parameter values which depending on the human skill would affect the A matrix. The model was trained, and the loss converged very quickly.

2) Results

During the testing part, when the value to be tested is within the range in which the parameters were chosen, the neural network worked perfectly. But if the input to the neural network was outside the range, the results were unexpected.

We plotted the controller output against changing parameter values in a 3d graph. Two graphs obtained for K (1,6) and K (2,6) was discontinuous in this range. The rest all were continuous. Also, it was found that the discontinuous ones are the values which are highly incorrect while testing with the neural network outside the training range.

As we were not sure about the range of values we were using for this system, we decided to simulate an inverted pendulum system to understand more.

We decided to keep the mass of the cart and the pendulum as changing parameters. The range varying between 1-100Kg. After doing the same process, we observed that the 3d graphs for this system were continuous. When we tested outside the range of training also, the neural network worked perfectly well.

In conclusion, this method can be used for systems which have changing parameters within a specified range. This is acceptable as practically the parameters won't exceed certain values. The paper mentioned in [2] from which we took reference, uses the same approach for varying the speed and flux state for an induction motor drive. As mentioned in the paper, the approach is used to solve a nonlinear and non-stationary plant control task with the help of linear time-invariant (LTI) system theory and ANN-based function approximator

VI. References

[1] H. Modares, I. Ranatunga, F. L. Lewis and D. O. Popa, "Optimized Assistive Human–Robot Interaction Using Reinforcement Learning," in *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 655-667, March 2016

[2] L. M. Grzesiak and B. Ufnalski, "Neural-Network-based Programmable State Feedback Controller for Induction Motor Drive," *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, Vancouver, BC, 2006, pp. 1091-1097