Learning Optimal Control

Kiran Kumar Sathyanarayanan   
*Masters Automation and Robotics*  
*Techniche Universität Dortmund*Dortmund, Germany  
kiran.sathyanarayanan@tu-dortmund.de

Rajesh Pushparaj  
*Masters Automation and Robotics*  
*Techniche Universität Dortmund*Dortmund, Germany  
rajesh.pushparaj@tu-dortmund.de Prathap Amudha Ramesh Babu  
*Masters Automation and Robotics*  
*Techniche Universität Dortmund*Dortmund, Germany  
prathap.amudha@tu-dortmund.de

Sriram Saravanan  
*Masters Automation and Robotics*  
*Techniche Universität Dortmund*Dortmund, Germany  
sriram.saravanan@tu-dortmund.de

*Abstract*—Double inverted pendulum is one of the most popular unstable and chaotic system, it has two joints and two rods attached to a moving cart. In this project we try to imitate the Model predictive controller with Feedforward Neural Network that makes the double inverted pendulum to stand upright and makes it stay in that position. And we also use Reinforcement learning to make a Neural Network that learns to make the pendulum stand upright using reward and action, then we compare the training complexity and performance of the controllers to determine the best one in terms efficiency.

Keywords—Imitation Learning, Reinforcement Learning, dataset, Model Predictive Control, Neural Network, Adam, ReLu, tanh, ddpg, actor-critic, Q-value, Loss function.

# Introduction (*Heading 1*)

Controlling unstable and chaotic systems has lot of applications in Automation, Robotics and Process Control. One such classic chaotic system is double inverted pendulum, it has two rods with two joints attached to a moving cart. The task is to make the pendulum stand upright without falling, we can model predictive controller(MPC) to achieve this task. Model predictive control is an advanced method of process control which uses the model of the system and initial state to predict the future of the systems using optimization algorithms. In this project we try to imitate the model predictive control by collecting dataset for different initial states and using the MPC to predict the force given to the cart for various state of the pendulum.

In imitation learning we use supervised learning to create a feedforward neural network and train it with datasets generated using the MPC controller. And after training we export the model and simulate the model with the same initial condition given for generating the dataset with MPC. And the exported model predicts the force similar to MPC does.

In the second part of the project we try to achieve the task using reinforcement learning(RL). Where the training of the neural network is done using reward and action. The process take the model as an environment and uses an agent which tries to achieve the desired task and a feedback is given to the agent using a critic which uses reward function to calculate the reward for each episode of training. After training the neural network(NN) is exported similarly as done in imitation learning and simulated to the initial state for which the NN is trained.

Finally we compare the training complexity and performance of the both imitation learning NN and the RL based NN model to determine the best model in terms efficiency for this double inverted pendulum problem.

The process of training NN model for each case is explained in detail in the upcoming sections.

# imitation learning

In imitation learning we use the MPC to control the pendulum making it stand upright and record the states and corresponding force in cart. We create a linear regression model for the recorded data and use it to control the pendulum thus imitating the MPC’s predictions. The process contains the following steps which are explained below

## Data generation

First, we start collecting the datapoints for various initial states of the pendulum using the MPC controller. In our system we have the following six states

*x, θ*1*, θ*2*, x*˙ *, θ*˙1*, θ*˙2

*x* will be the position of the cart, *θ*1 and *θ*2 are the angle of the first and second pendulum, *x*˙ is the cart velocity and *θ*˙1 *a*nd *θ*˙2 are the angular velocity of the pendulums.

For various initial states we try to make the pendulum stand upright using the MPC controller and the data is collected as pair of states and force to the cart. In order to improve the dataset for each initial state, MPC controller is ran for 4 times for each initial states.

The data files are converted to .xls format for easy access using the pandas library. The datasets are then checked for duplicates and null entries and these data is filtered. Then the data is scaled in order to improve the learning rate of the training.

Then the data is split into two sets test and training randomly. The train set is used for training the model and the test data is for evaluating the model after training. We have split the data into 80% of train data and 20% of test data. Now, this data is used for training the model.

## Neural Network Architecture

Next step is to create a neural network for learning from the datasets. The architecture of the neural network is as follows,

Input layer should have 6 nodes for the 6 states for input. Output layer will be one since we have only one output force for the cart. There are 6 hidden layers with 64, 64, 128, 128, 256, 256 nodes in each hidden layer.

We have no activation function for the input and the output layer and all the other layers have *tanh* as the activation function. *Adam* is used as the optimization algorithm with the learning rate 0.01. Mean Squared Error is used as the loss function.

We use keras framework from tensorflow with python for building and training the model. The model is initially compiled with random weights.

## Model building and simulation

For the training the model training is ran for 1000 epochs and the callback function is used to save the best weights throughout the training process.

The training is done till the loss function reaches 0.2 and the model and the weights are exported for simulation.

The model and weights exported from the training is used to predict the force of the cart for the states. The initial state is taken from one of the initial state used for generating the datasets. And the NN model is used in place of the MPC to predict the force. The results are rendered in ImageMagick tool.

# reinforcement learning

Reinforcement learning is the training of machine learning models to make a sequence of decisions. RL has the following components agents, environments, states, actions and rewards.

RL models are given an environment, a set of actions they can perform, and a goal or a reward they must pursue. Agents act on the environment and try to maximize the cumulative reward so that it will learn to do the given task.

In this project we use the double pendulum environment and make the agent to learn to make the pendulum to stand upright by taking actions with the given feedback. The entire process in explained in detail as follows,

## Double pendulum environment

First, we should have the environment of our system for the agent to act upon and learn. We used *gym* library from OpenAI to generate the environment for the double inverted pendulum.

A *gym* environment should have 4 important elements init, step, reset and render.

Init – It contains the action space, observation space and the environment parameters such as mass of the cart and poles, length of the poles, etc.

Step – It computes the rewards and new states based on the input state using the model of the pendulum.

Reset – It reset the state of the pendulum to the initial position after each episode. Episode is one learning cycle in RL.

Render – It is used to visualize the learning process.

Init function contains the system parameters, angle at which the episode has to fail, and initialize the observation space. Step function has the model of the system itself, cost function for reward. Here, the potential energy is used to calculate the reward.

Add cost function here

And it computes the new state based on the input and action. Reset function brings the pendulum back to initial state for the new episode.

## DDPG parameters

Deep Deterministic Policy Gradient is a model free off-policy RL algorithm for learning continuous action. Since our system is a continuous system DDPG is the best option for this task. It has experience replay and slow-learning target networks.

Experience reply enables reinforcement learning agents to memorize and reuse past experiences, just as humans replay memories for the situation at hand.

There are two NN Actor and critic, Actor takes the state as input and outputs the best action based on the value function and the critic takes the state and reward as the input and predicts the best set of action which yields the maximum reward using the Q-function.

Actor NN has the following architecture, Input layers with 6 nodes, 2 hidden layers with 256 nodes and 1 output layer. Hidden layers has *ReLu* as activation function and output layer has *tanh.* Adam optimization algorithm with 0.0005 learning rate is used.

Critic NN has the 2 NN for state and reward and finally concatenated to one NN. State NN has the architecture, 6 nodes in the input layer, 1 hidden layer with 16 nodes and output layer with 32 nodes. Reward NN has the architecture of 1 node in the input layer and 32 nodes in the output layer. Both have *ReLu*  activation function in the hidden layer and the output layer has *tanh*. It has Adam optimization algorithm with 0.002 learning rate.

Actor will have less learning rate compared to critic since actor should choose the action trajectory based on the critic’s Q-function value.

The buffer size for the experience reply is kept as 100000, batch size is 128, Gamma value is 0.99 and the target update rate tau is 0.01.

## Algorithm

Add the ddpg algorithm pseudo code here

The learning process is very closely related to Q-learning where if you know the optimal-action-value function Q\*(s, a)

DDPG was developed specifically for dealing with environments with continuous action spaces and in essence that is to estimate the max over actions in **max Q\*(s, a).** Best and optimal action to taken in that state can be found out using a\*(s),

Add a\*(s) equation

In the case of Continuous action spaces, computation and individual comparison for each Q-value becomes very exhaustive leading to non-stationary target values and unstable learning. Not to mention, the process for such is quite exhaustive and computationally expensive.

Q-learning based algorithms, specifically DDPG employs the use of the following to deal with a continuous action space:

Make use of the Bellman equation to obtain the optimal action for a given state using its state-action/Q-value.

Add bellman equation here

DDPG employs the use of mean-squared Bellman error (MSBE) function which estimates how close Q\* comes close to satisfying the Bellman equation as shown in the equation:

Add Mean Squared Bellman Error value equation

Making use of Experience Replay Buffer which is a set of previous experiences which helps in providing Q-learning based approximators a stable learning behavior.

DDPG also deploys the use of a Target network to deal with non-stationary target values and make the learning more stable. Following describes what a Target is because when we minimize the MSBE loss, we are trying to make the Q-function be more like this target.

Add Target Value

DDPG’s target network which is just copied over from the main network some-fixed-number of steps is updated once per main network update by Polyak averaging

Add polyak averaging

Thus DDPG deals with this humongous continuous action space challenge and expensive computation by using a target policy network to compute an action that approximately maximizes Q\*(Target).

## Learning process

After setting all the above parameters for learning, training is started with initializing the pendulum in a random initial state.

For every episode the agent tries to maximize the reward and fails when the pendulum moves away from the failing point. Failing results in a negative reward and the agent learns from this and tries other approaches to increase the reward and thus learning better every episode.

Target network is updated for every optimal reward thus keeping the best possible weights stored in the network. Target network helps to stabilize the learning process and thus making the learning more efficient and robust.

The learning is done for 1000 episodes for more precise learning. And tries to maximize the cumulative reward. And the end of the training process both action-critic and the target networks should have the same or almost same weights in the Neural Network.

The action NN model and weights are exported as json and h5 files and imported in the simulation environment which uses the same initial state as that of the training. The NN tries to predict the forces required for the cart to make the pendulum stand in the upright position.

Add the training image here

# comparison

After training NN using Imitation learning and Reinforcement learning we compare the training complexity and performance of the both to determine the efficient method.

For imitation learning we used 17,692 data after filtering taken from 62 datasets for different initial states. This method relies on the quality of the datasets, having dataset that have proper state and force value pairs for all possible states in the process is required for proper training of the model. Acquiring this huge amount dataset is time consuming and need high computational power. But after the model is generated using it as a controller is less burden for the device and easy to use.

For Reinforcement learning since it is a model free machine learning method no dataset is needed for training the model. Challenge lies in creating the environment from scratch since the *gym* environment for the double inverted pendulum isn’t a readily available one. We used the single inverted pendulum environment as a base and modified the source code to create the environment which programmatically challenging. For a better learning the agent has to run approximately 1000 episodes while training and it’s a huge burden for the device to run two computationally intensive optimization algorithms every episode for 1000 times which makes RL more computationally intensive than the imitation learning method.

Comparing the performance of the two models, both the methods struggled to make the pendulum stay in the upright position but whereas Reinforcement learning had better performance in making it go to the upright position and tried better to stay in that position. In order to make the Imitation learning to achieve the same performance lot of data pre-processing to filter the best data pairs from the MPC controller is needed and its very time consuming and there is no easy way doing it.

Better results may be acquired by combining the two methods and using it to control the pendulum.

##### Acknowledgment

We would like to express our gratitude to Professor Dr Sergio Lucia and all the tutors who guided us throughout this project. We would also like to thank my friends who supported me and offered deep insight into the study.

##### References

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**