First order system control with Artificial intelligent

Reinforcement Learning algorithm

Deep Q-learning algorithm (DQN)

Using MATLAB 2019b

Script:

%%%%%%%%%%%% Environment Creation %%%%%%%%%%%%%

mdl = 'FirstOrderSystemSimulinkRL';

open\_system(mdl)

obsInfo = rlNumericSpec([3 1]) % vector of 3 observations: e de/dt integral of(e)

actInfo = rlFiniteSetSpec([-1 0 1]) % 3 actions

obsInfo.Name = 'observations';

actInfo.Name = 'Action';

agentBlk = [mdl '/RL Agent'];

env = rlSimulinkEnv(mdl,agentBlk,obsInfo,actInfo)

Ts = 0.05;

Tf = 20;

rng(0)

%%%%%%%%% Neural Network %%%%%%%%%%%

statePath = [

imageInputLayer([3 1 1],'Normalization','none','Name','state')

fullyConnectedLayer(24,'Name','CriticStateFC1')

reluLayer('Name','CriticRelu1')

fullyConnectedLayer(48,'Name','CriticStateFC2')];

actionPath = [

imageInputLayer([1 1 1],'Normalization','none','Name','action')

fullyConnectedLayer(48,'Name','CriticActionFC1','BiasLearnRateFactor',0)];

commonPath = [

additionLayer(2,'Name','add')

reluLayer('Name','CriticCommonRelu')

fullyConnectedLayer(1,'Name','output')];

criticNetwork = layerGraph();

criticNetwork = addLayers(criticNetwork,statePath);

criticNetwork = addLayers(criticNetwork,actionPath);

criticNetwork = addLayers(criticNetwork,commonPath);

criticNetwork = connectLayers(criticNetwork,'CriticStateFC2','add/in1');

criticNetwork = connectLayers(criticNetwork,'CriticActionFC1','add/in2');

figure;

plot(criticNetwork);

%%%%%%%%%%%% RL Options %%%%%%%%%%%%%%%%%

criticOptions = rlRepresentationOptions('LearnRate',0.001,'GradientThreshold',1);

critic = rlRepresentation(criticNetwork,obsInfo,actInfo,...

'observation',{'state'},'Action',{'action'},criticOptions);

%%%%%%%%%%%% RL-DQN Configuration %%%%%%%%%%%%%

agentOptions = rlDQNAgentOptions(...

'SampleTime',Ts,...

'TargetSmoothFactor',1e-3,... %target network updates method

'ExperienceBufferLength',1e6,... %replay memory length

'UseDoubleDQN',false,... %Double Deep Q-learning algorithm is disabled

'DiscountFactor',0.99,... %discount factor

'MiniBatchSize',64); %mini batch for replay memory

agent = rlDQNAgent(critic,agentOptions);

%%%%%%%%%%%%%%% Training Options %%%%%%%%%%%%%%%

trainingOptions = rlTrainingOptions(...

'MaxEpisodes',200,...

'MaxStepsPerEpisode',500,...

'ScoreAveragingWindowLength',10,...

'Verbose',false,...

'Plots','training-progress',...

'StopTrainingCriteria','AverageReward',...

'StopTrainingValue',100,...

'SaveAgentCriteria','EpisodeReward',...

'SaveAgentValue',100);

%%%%%%%%%%%%%%% Training %%%%%%%%%%%%%%%

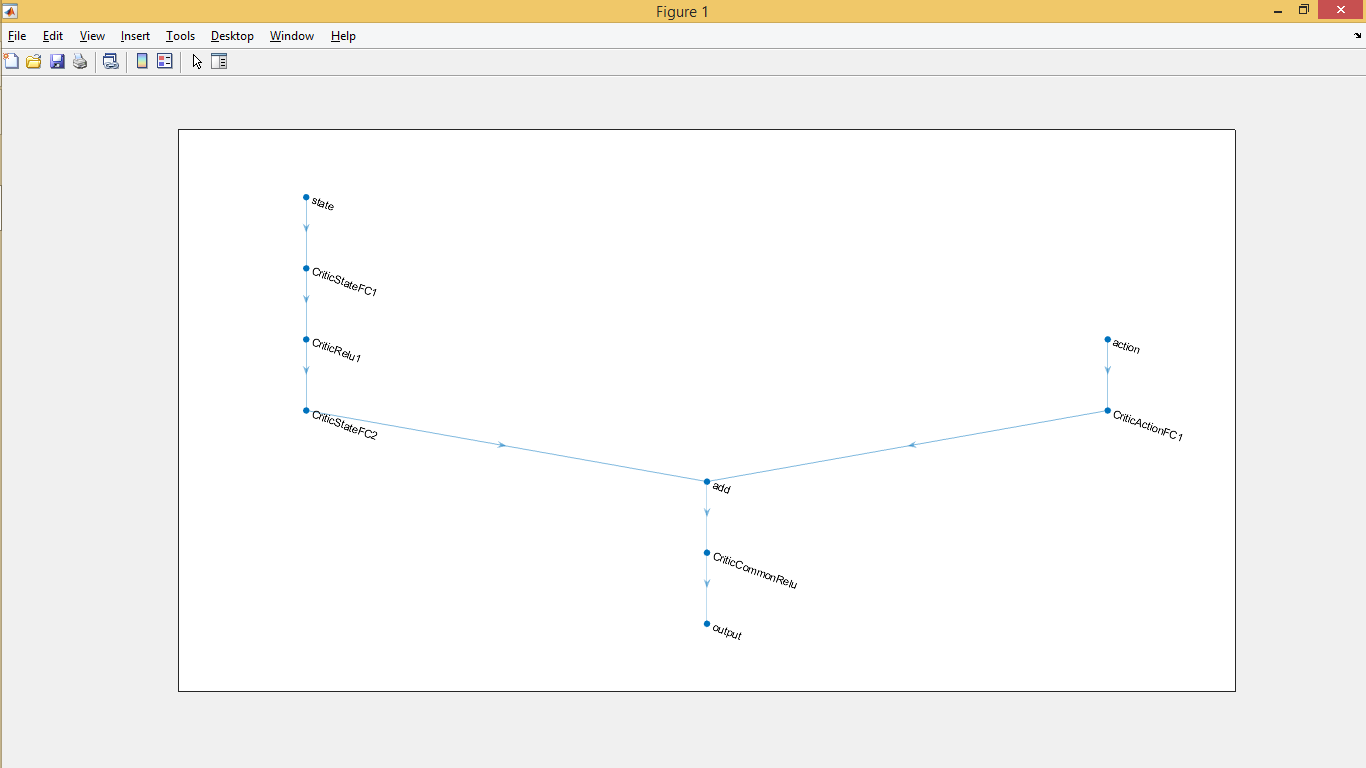
trainingStats = train(agent,env,trainingOptions);

%%%%%%%%%%%%%%% Simulation %%%%%%%%%%%%%

simOptions = rlSimulationOptions('MaxSteps',500);

experience = sim(env,agent,simOptions);

Neural Network graphic representation:



Simulink control:

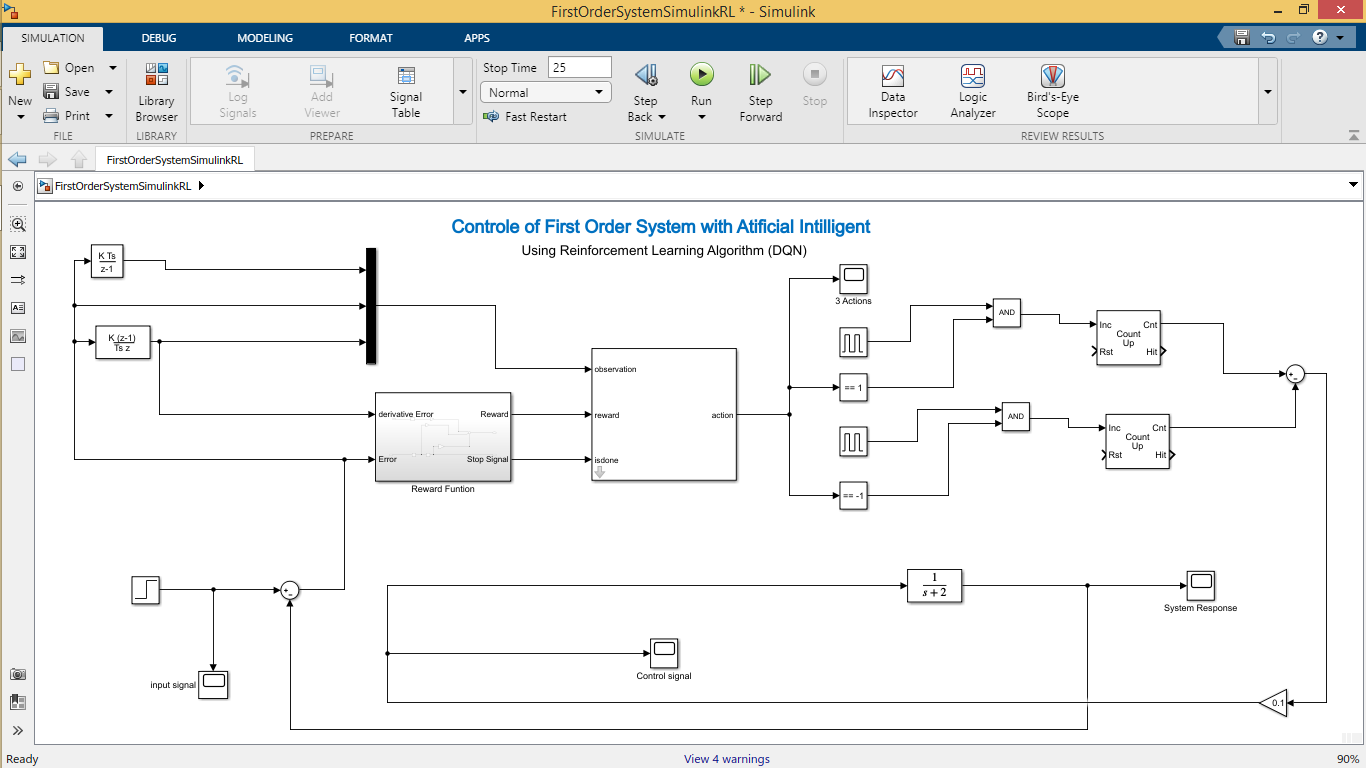


Figure1: Simulink simulation

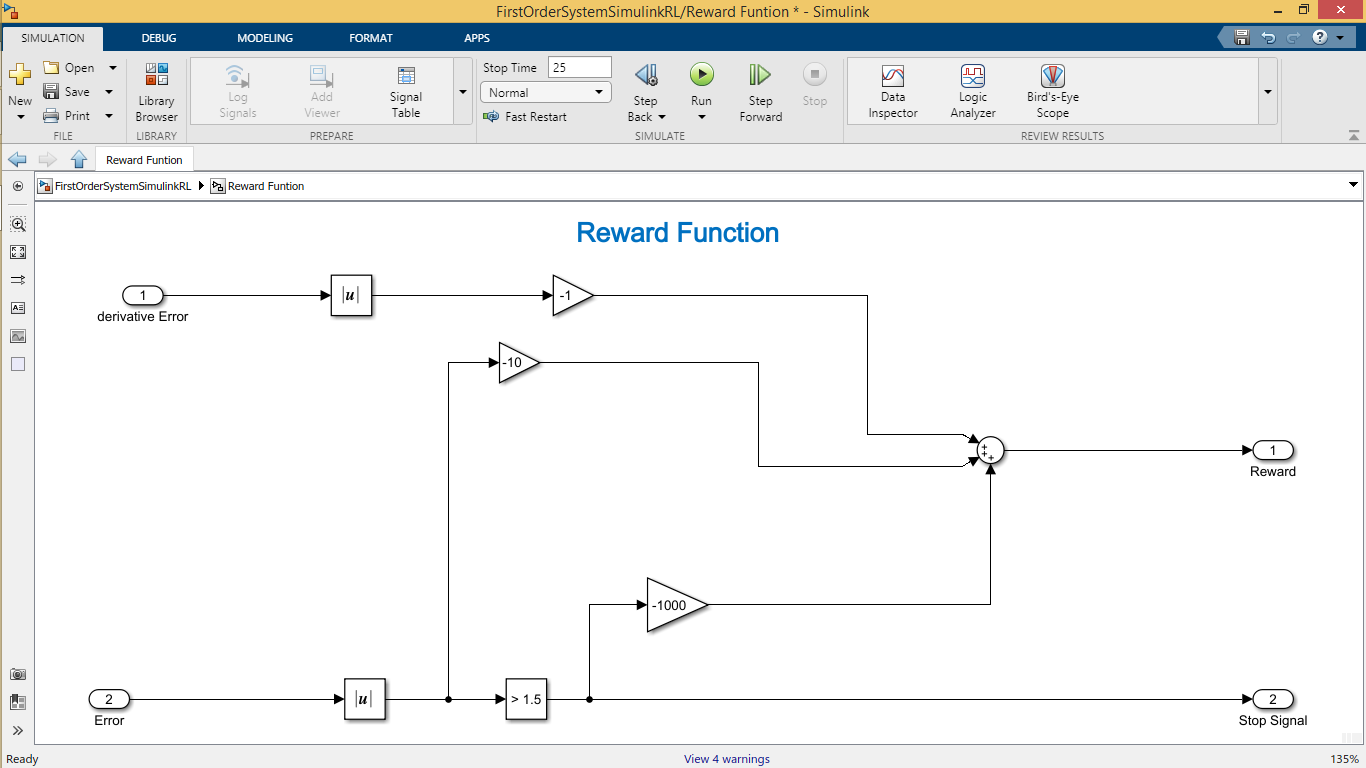
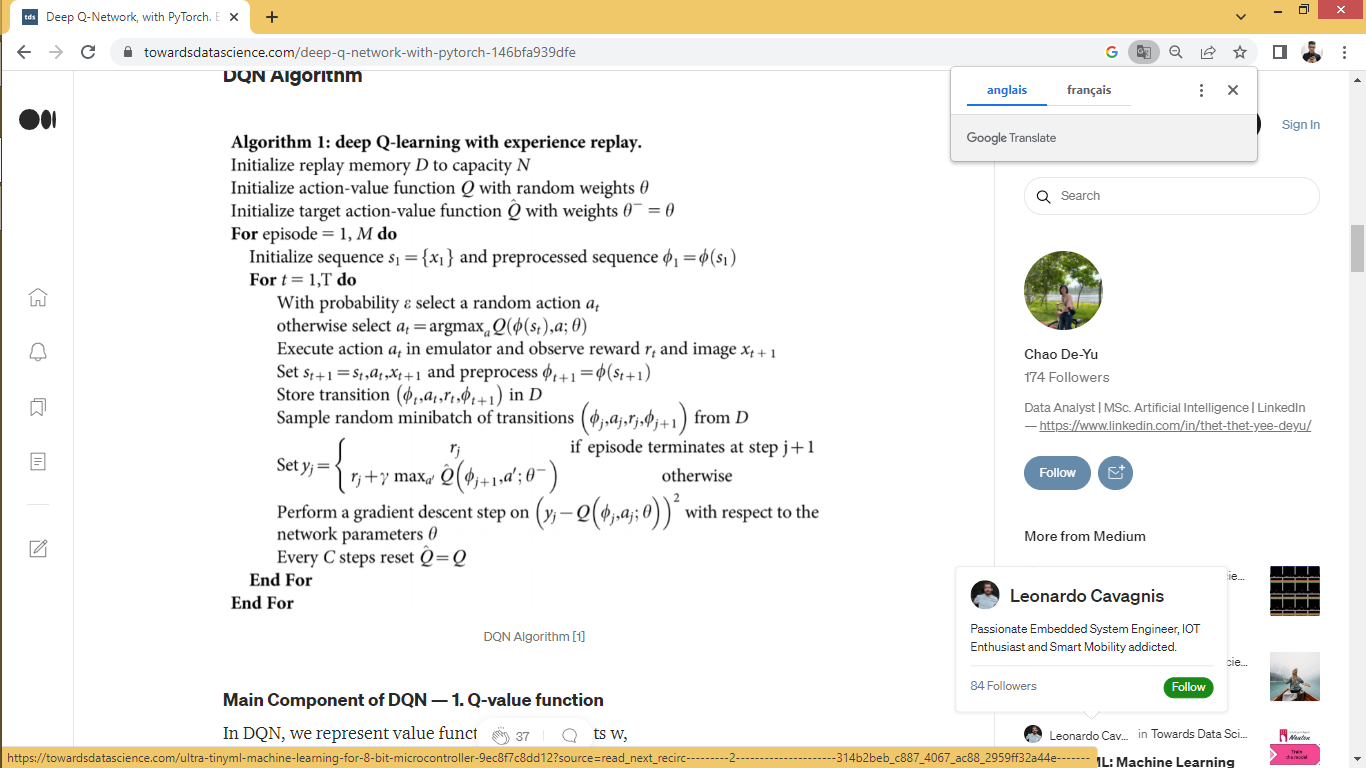


Figure2: Simulink reward function

\_Algorithm:



* **Training:**

We trained this model on a random Inputs between zero and one randomly and for 200 episodes.

The model trained by Deep Q-learning algorithm. Its outputs are discrete (actions space). in this example we have three actions control an incremental and decremental system with counters (+1 action increment, -1 action decrement, 0 action do not do anything), with high frequency.

* **Reward function:**

The reward function represents the goal of the experience. It takes the derivative of error as negative (-1) in each step, it decreases the variations and vibration of the system response. The second function is the error with a negative (-10) in each step to guaranteed the precision of the system response. The last reward is a negative (-1000) to avoid any exceeding of error (>1.5).

* **Observations:**

For more inputs training information, we toke the error and the derivative of error and the integral of error as input for the RL model.

* **Neural network:**

Medium size of Neural Network, because of simplicity of the environment (system), for complex systems we need larger Neural Network.

* **Conclusion:**
* Control without having a model for the system.
* Train a model to achieve our goal. (Maximizing the reward function)
* the downsides of RL that we don’t a mathematical or physical representation of the RL-Model (controller), so we can’t do any prediction or studies of its performances.

Test01: Step input (Final value = 1)

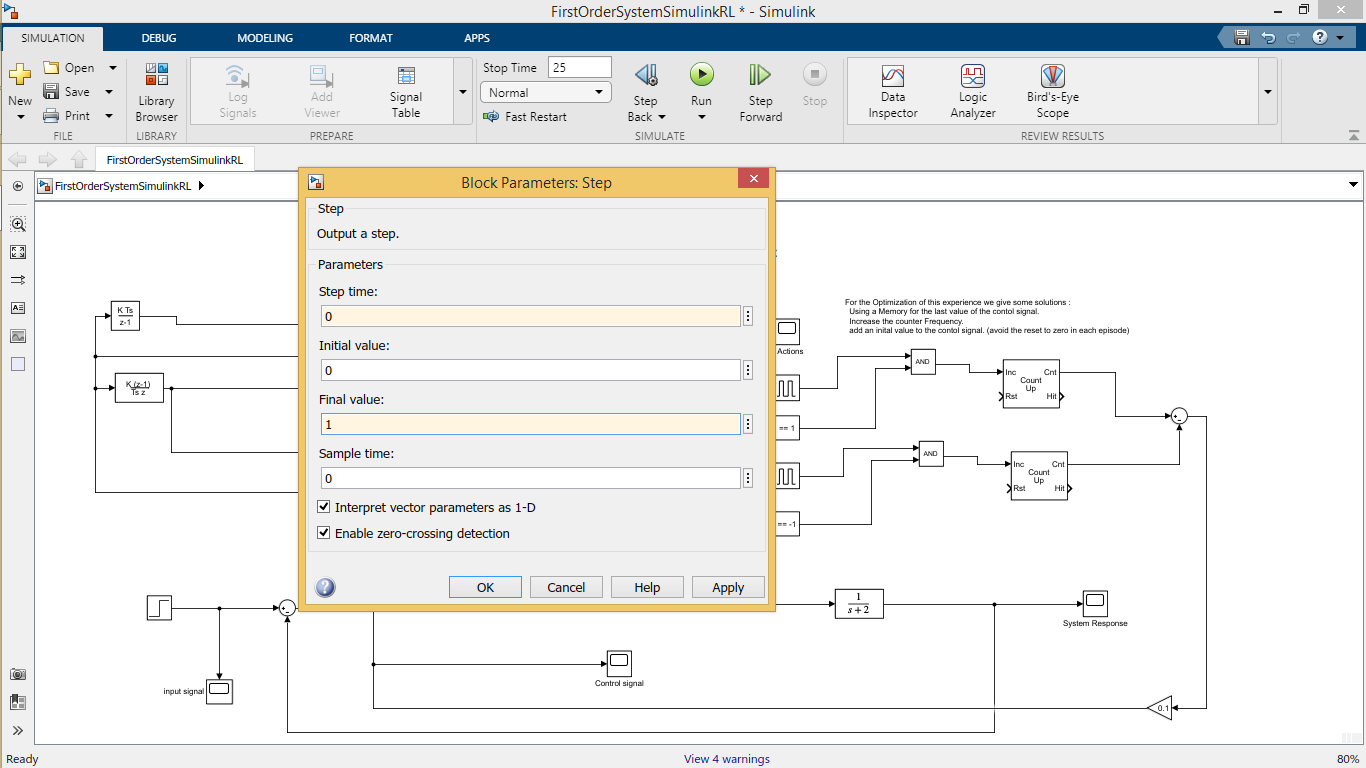


Figure1: Step Block option



Figure2: input signal

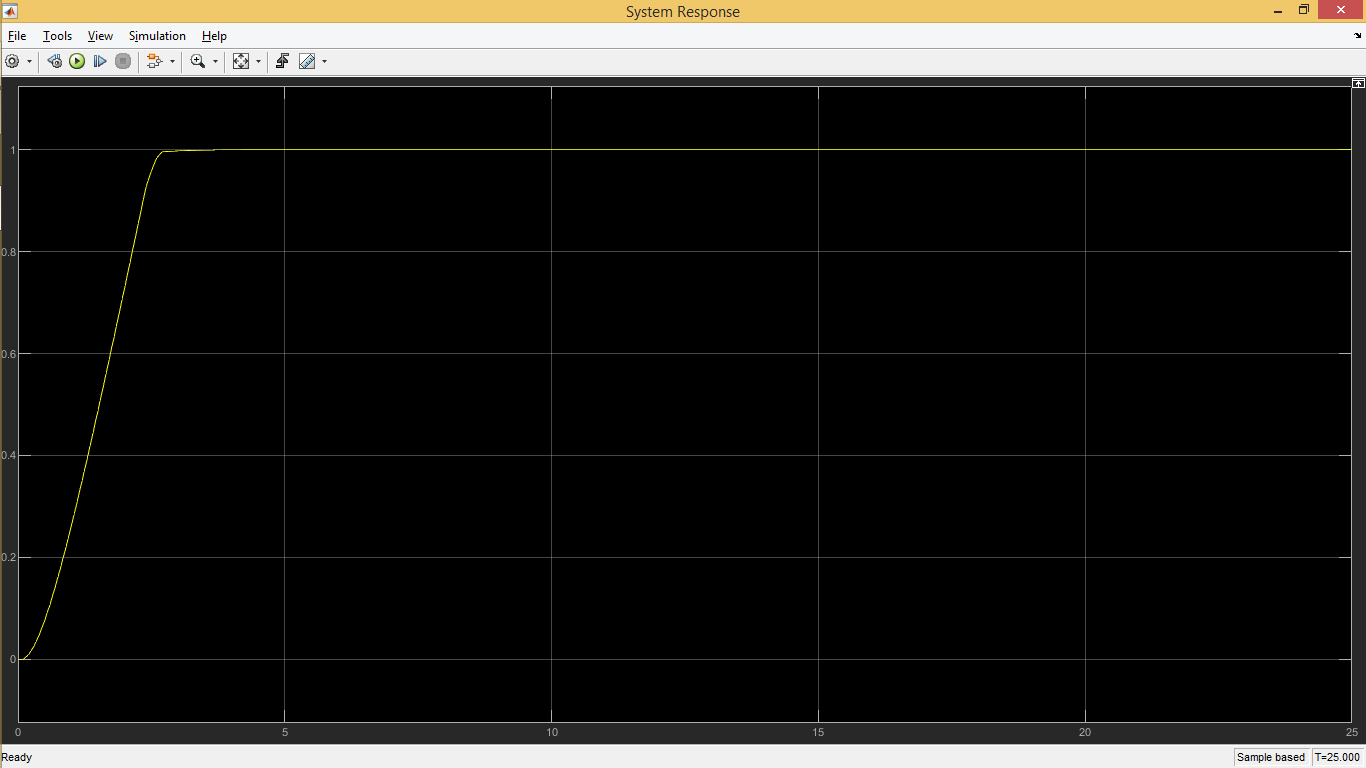


Figure3: system response



Figure4: output of RL model (3 actions)



Figure5: control signal

Test02: Step input (Final value = 0.2)

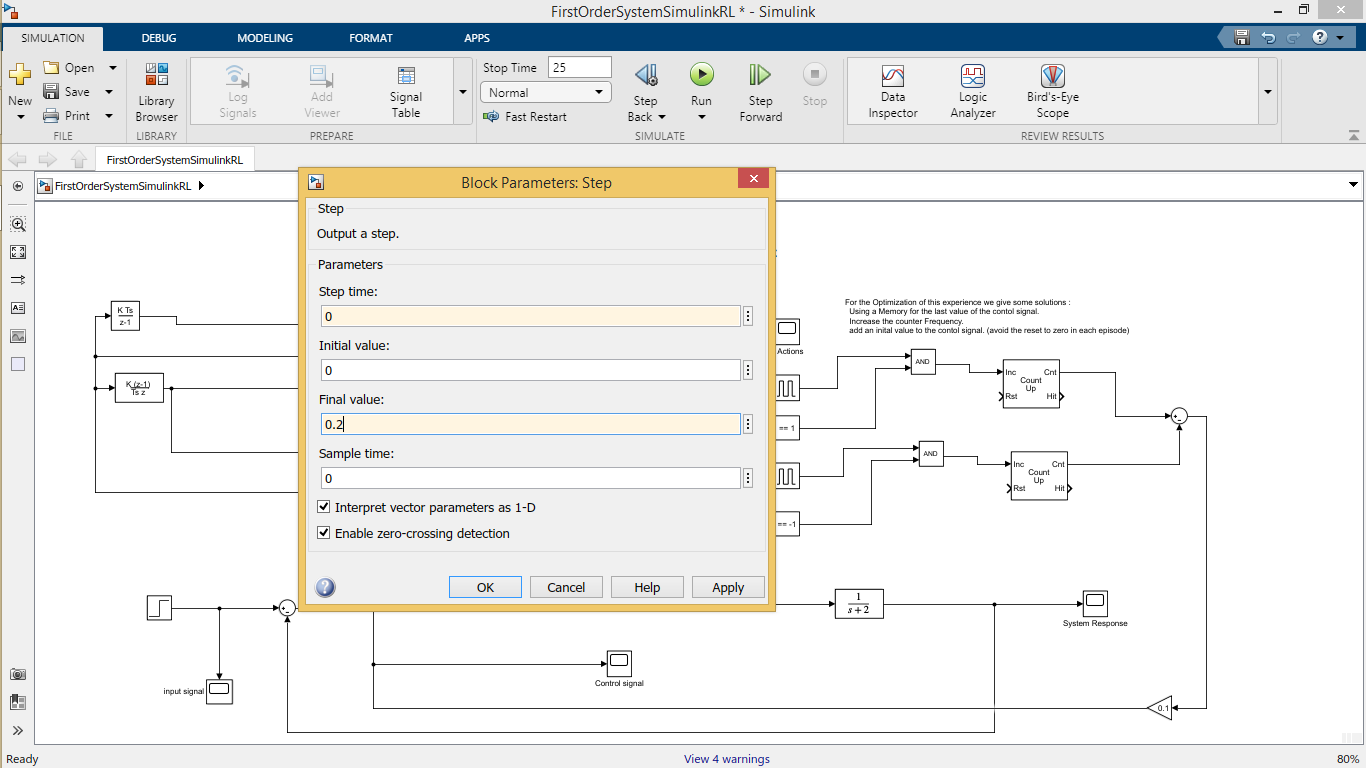


Figure1: Step Block option

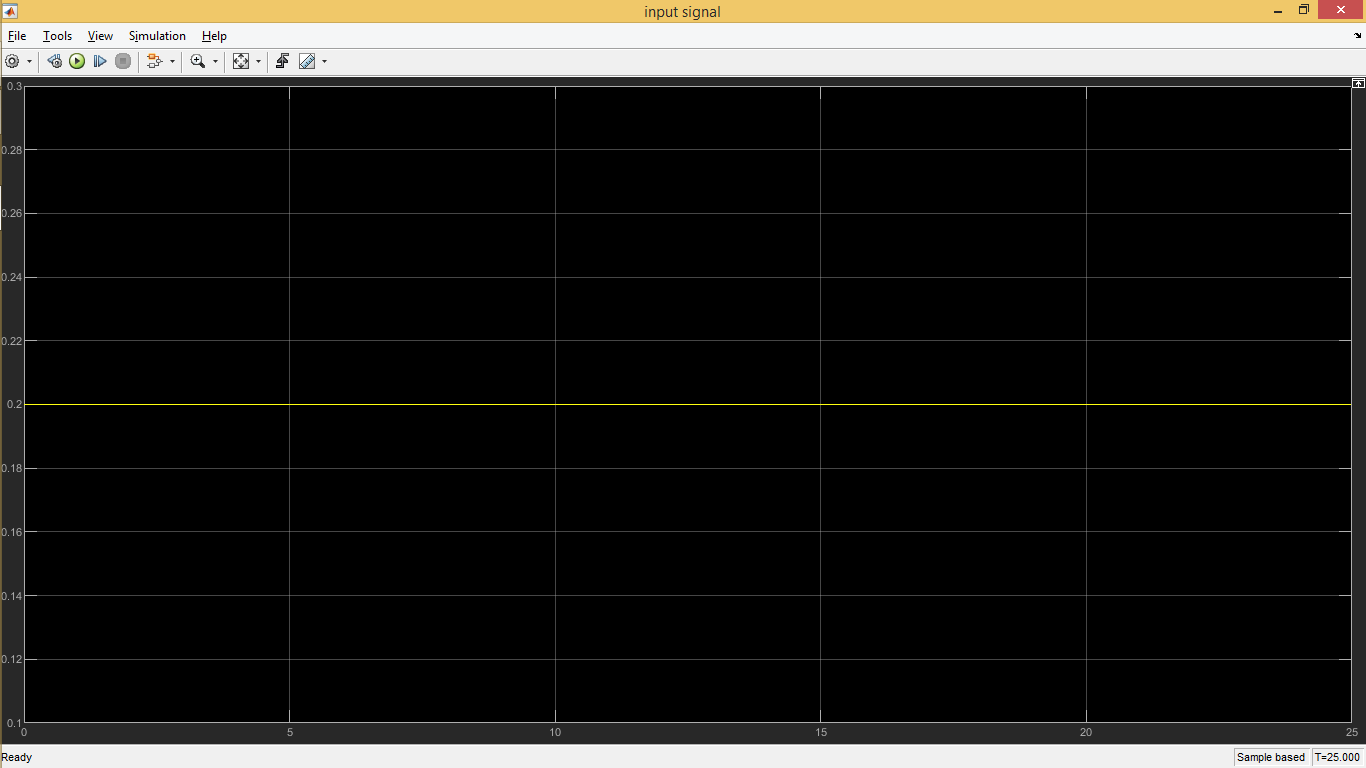


Figure2: input signal



Figure3: system response



Figure4: output of RL model (3 actions)

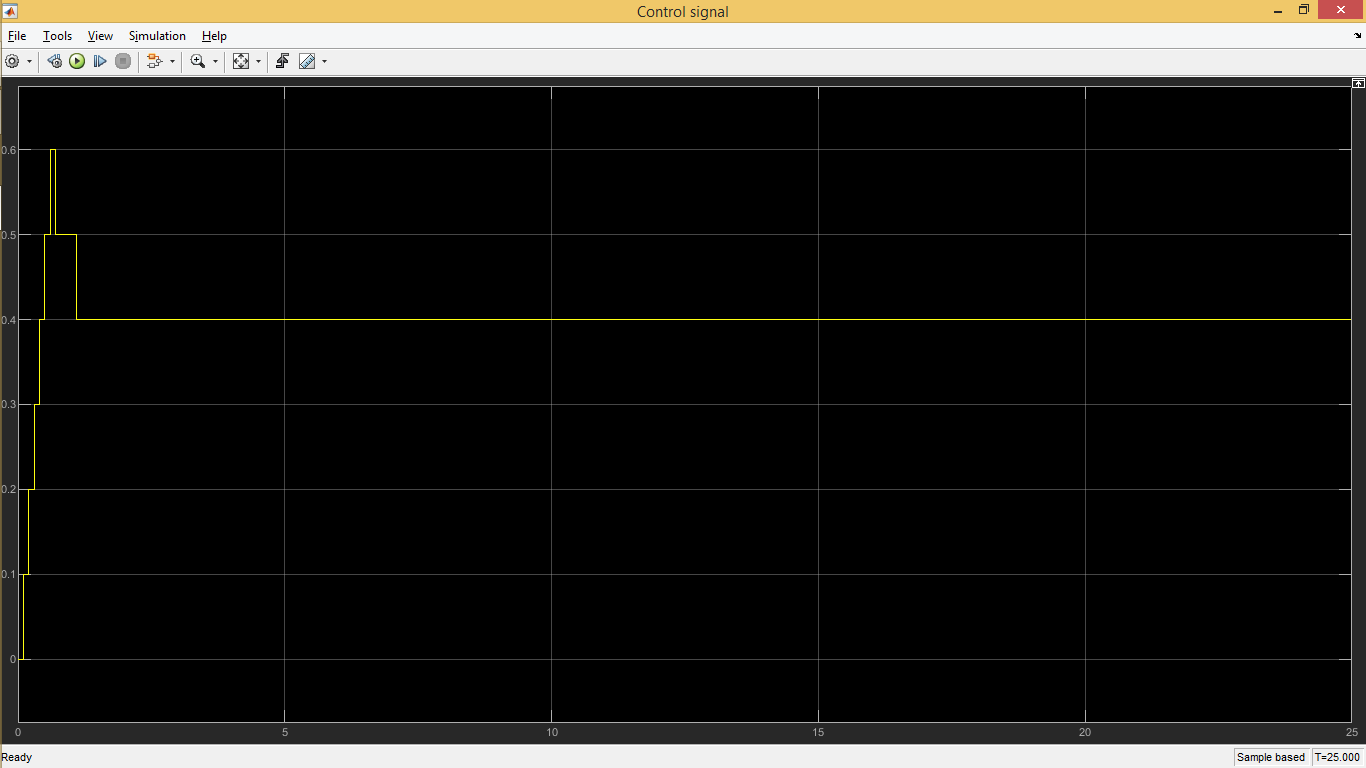


Figure5: control signal

Test03: Step input (Final value = 0.7, step time = 2)

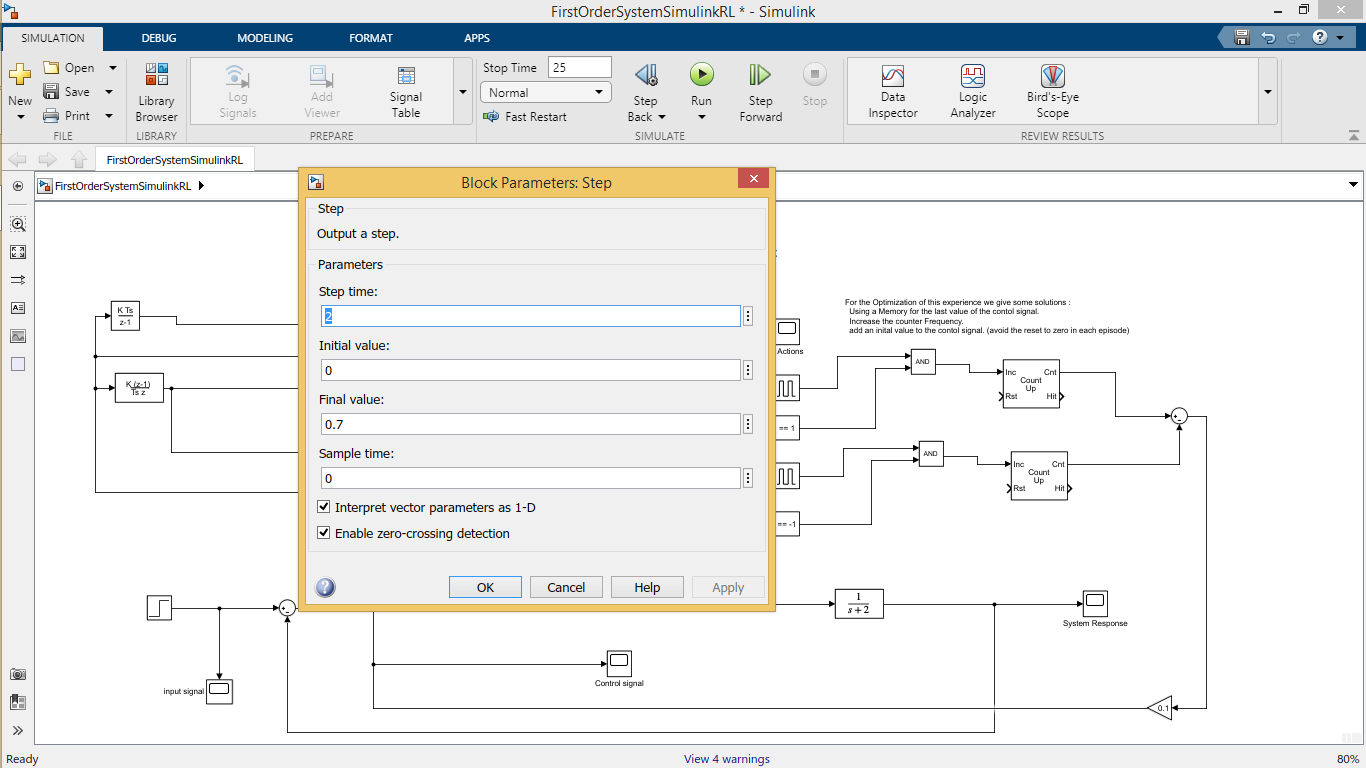


Figure1: Step Block option

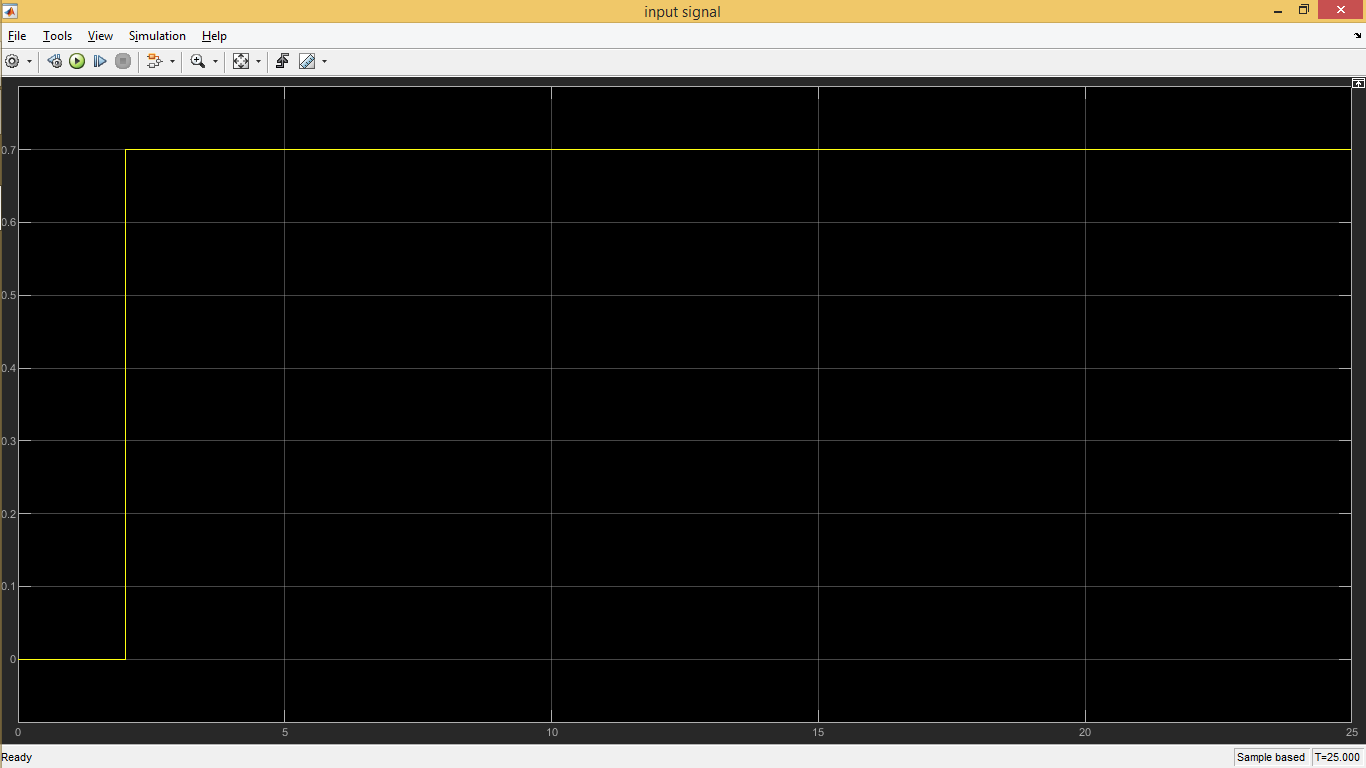


Figure2: input signal

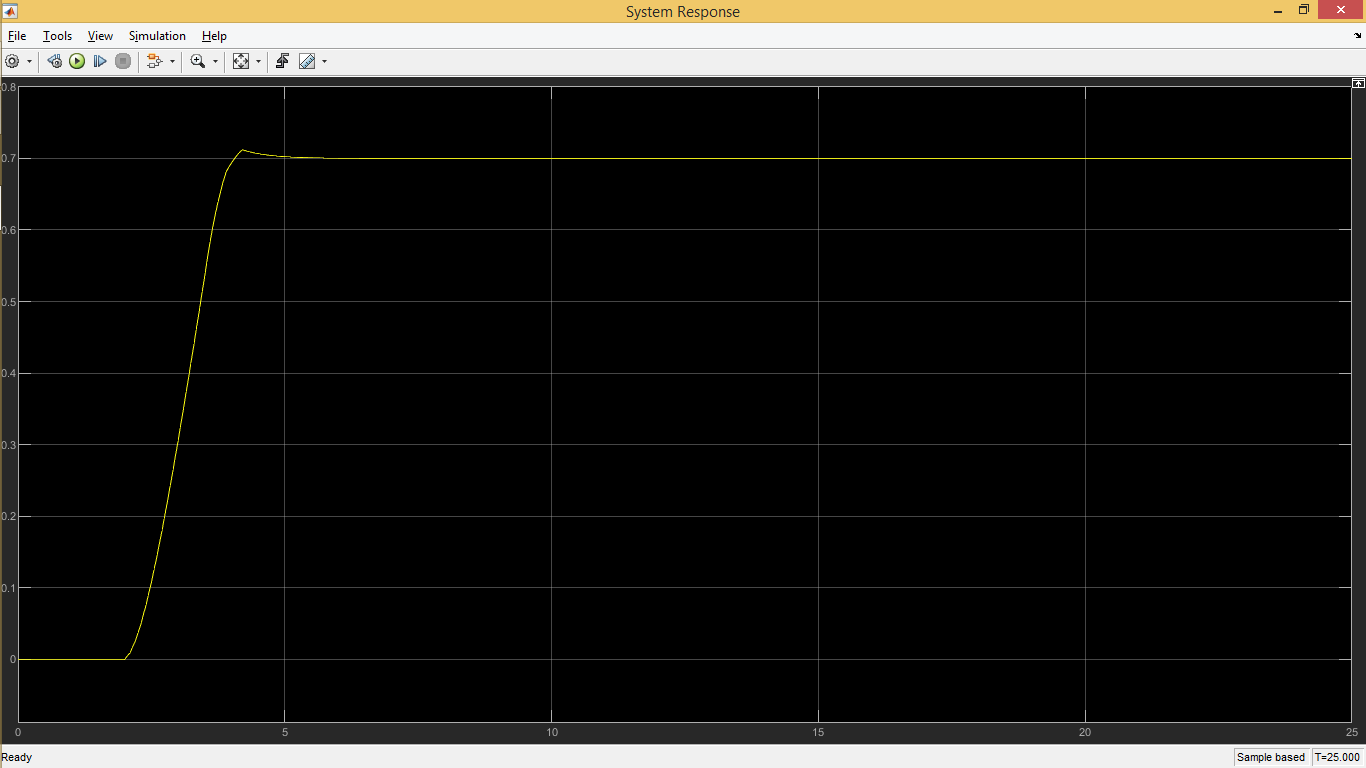


Figure3: system response



Figure4: output of RL model (3 actions)

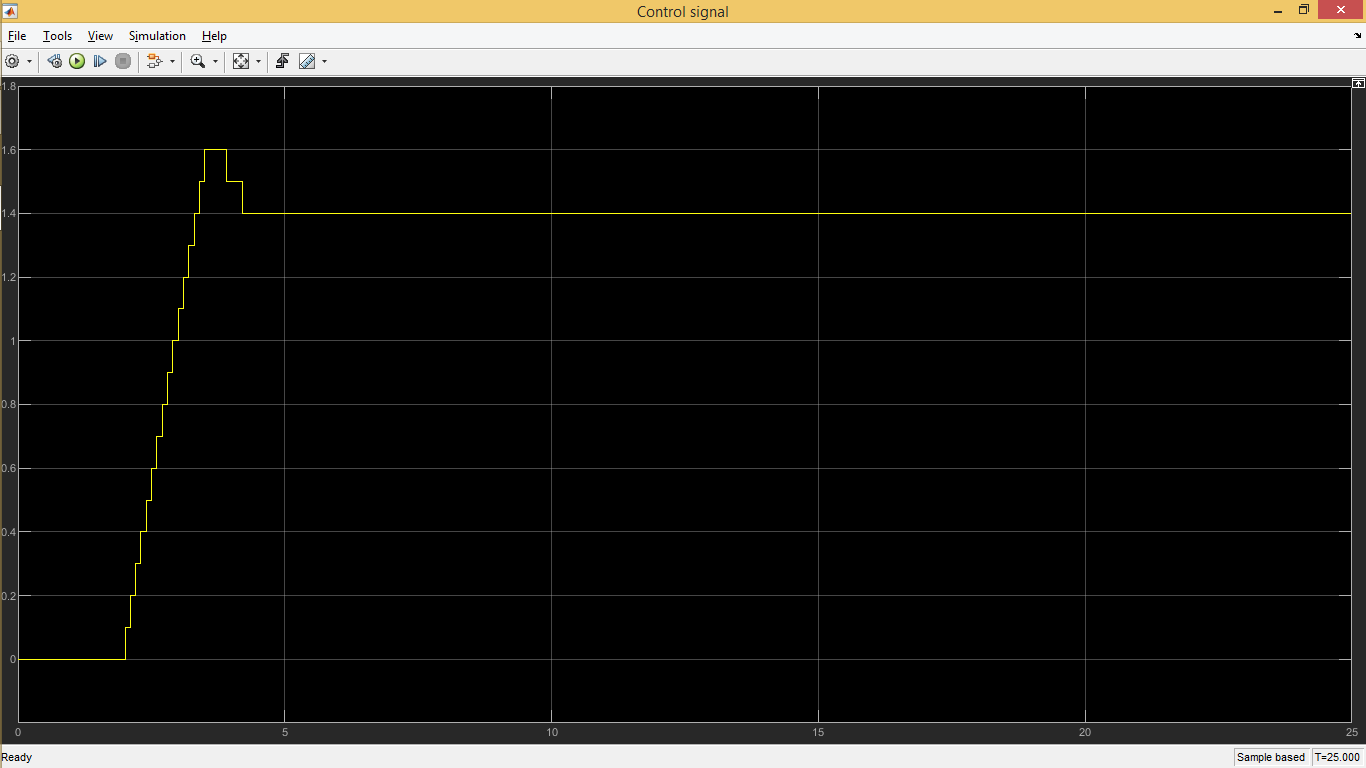


Figure5: control signal

Optimization Options:

\_Increase the counter frequency.

\_Add an initial value to the control signal. (Avoid reset to zero in each episode).

\_Use a memory for the last value of the control signal.

\_Train the model for more long Time and more episodes.

\_Decrease the discount factor to make the system faster

\_Increase the range of training inputs and the neural network hidden layers size.