A research on deep reinforcement learning in manipulator

Ming Lin

Abstract—Recently, reinforcement learning is a hot topic in machine learning. The network like CNN algorithm is a kind of supervised learning with trained data. It has input data and correspond labeled data. The inner relationship is learned by neural it work it self by mapping predicted data value and ground truth value. Reinforcement learning(RL) is a kind of different from above supervised learning. RL is a unsupervised learning algorithm. It receives input data and based on the reward function to optimizing the reward. So the reward function is very important for reinforcement learning. Classic algorithm in reinforcement learning is Q learning. It based on the markov decision process and reward table for getting highest reward from agent. In this research deep reinforcement learning(DRL) algorithm is implemented. Deep reinforcement learning algorithm is based on the multi neural network, and try to learning how to get highest reward from agent. The agent is created by using the Pytorch which is a python machine learning library. The action and sensor parameters are simulated by using the GAZEBO in ROS. Based on the simulation results, it can be found that DRL control the manipulator well and meet the requirements in udacity.

Index Terms—Robot, Manipulator, Udacity, LaTeX, DQN, Q learning, OpenAI Gym.

1 Introduction

Machine learning algorithm is a powerful tool in recent different kinds of research. Machine learning algorithm is divide in to two kinds. One is supervised, another one is unsupervised. Supervised learning is proved as a good algorithm. However, it performance is depends on data input. Sometimes, if there are no appropriate data to feeding, then supervised learning can't be used. So the unsupervised learning algorithm is needed. Deep Q learning is a kind of application of reinforcement learning. It based on the deep neural network and user defined reward function, try to get highest reward from environment. The reinforcement learning algorithm is based on the pytorch library and interface with Gazebo with Gazebo API. This research try to implement DQN in manipulator control algorithm.

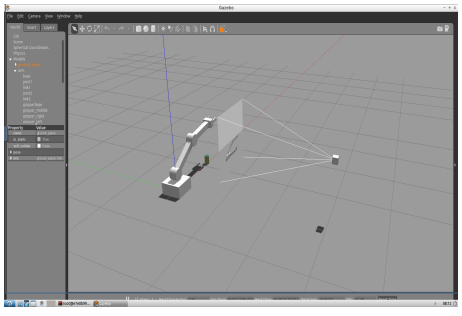

Fig. 1. Arm training in GAZEBO

2 RL Agent

Pytorch based reinforcement learning agent is created.

```
// Create DQN Agent
agent = dqnAgent::Create(INPUT_WIDTH, INPUT_HEIGHT, INPUT_CHANNELS, DOF*2,
                         OPTIMIZER, LEARNING_RATE, REPLAY_MEMORY, BATCH_SIZE,
                         GAMMA, EPS_START, EPS_END, EPS_DECAY,
                         USE_LSTM, LSTM_SIZE, ALLOW_RANDOM, DEBUG_DQN);
```

Fig. 2. agent create

3 Hyperparameters tuning

Hypterparameters are tuned as below figure. There are no significant differences between objective 1 and objective 2. Objective 1 only needs any part of manipulator contact with item, therefore the collision area of manipulator and item is pretty large. However, objective 2 needs more precise distance control, it needs more accurate collision area definition. The hyperparameters are defined as below.

1) Optimizer : Adam
2) Learning rate : 0.15
3) INPUT WIDTH, INPUT HEIGHT : 64
4) LSTM SIZE : 256
5) BATCH SIZE : 32

```
// Turn off velocity based control. Activate position control algorithm
#define VELOCITY_CONTROL false
#define VELOCITY_MIN -0.2f
#define VELOCITY_MAX  0.2f

// Define DQN parameter Settings
#define INPUT_CHANNELS 3
#define ALLOW_RANDOM true
#define DEBUG_DQN true
#define GAMMA 0.9f
#define EPS_START 0.9f
#define EPS_END 0.05f
#define EPS_DECAY 200

// Hyperparameters setting. Adam optimizer is used
#define INPUT_WIDTH   64
#define INPUT_HEIGHT  64
#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.15f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 256
```

Fig. 3. Define hyperparameters

4 REWARD FUNCTIONS

In order to control manipulator, reward functions should be defined appropriately. So it also need to decide to using position control method or velocity based method. Since velocity control need extra parameters, so the position control method is used. Obviously, position control is more simple than velocity control method. Subscribe to the collision sensor from gazebo world.

```
// Create our node for camera communication
cameraNode->Init();
cameraSub = cameraNode->Subscribe("/gazebo/" WORLD_NAME "/camera/link/camera/image", &ArmPlugin::onCameraMsg, this);

// Create our node for collision detection
collisionNode->Init();
collisionSub = collisionNode->Subscribe("/gazebo/" WORLD_NAME "/" PROP_NAME "/tube_link/my_contact", &ArmPlugin::onCollisionMsg, this);
```

Fig. 4. Subscribe collision sensor nodes

Parameters of reward are set as below.

1) REWARD WIN : 100.0
2) REWARD LOSS : -100.0
3) REWARD MULTIPLIER 10.0
4) Collision area which is named 'COLLISION ARM' is added.

```
// Define Reward
#define REWARD_WIN   100.0f
#define REWARD_LOSS -100.0f
#define REWARD_MULTIPLIER 10.0f

// Define Object Names
#define WORLD_NAME "arm_world"
#define PROP_NAME  "tube"
#define GRIP_NAME  "gripper_middle"

// Define Collision Parameter. Add a new COLLISION_ARM part from gazebo.world
#define COLLISION_FILTER "ground_plane::link::collision"
#define COLLISION_ITEM   "tube::tube_link::tube_collision"
#define COLLISION_POINT  "arm::gripperbase::gripper_link"
#define COLLISION_ARM    "arm::link2::collision2"

// Animation Steps
#define ANIMATION_STEPS 1000

// Set Debug Mode
#define DEBUG false

// Lock the base link
#define LOCKBASE true
```

Fig. 5. define parameters for reward

In order to give a appropriate reward, reward is given with a proportion with 'dist goal' parameter. If arm contacts ground, then assigns a negative reward.

```
if( gripBox.min.z <= groundContact || gripBox.max.z <= groundContact )
{
    // set appropriate Reward for robot hitting the ground.
    printf("GROUND CONTACT, EOE\n");

    rewardHistory = REWARD_LOSS;
    newReward     = true;
    endEpisode    = true;
}
else
{
    // for interim reward based on the distance to the object
    const float distGoal = BoxDistance(gripBox, propBox); // compute the distance from propbox to the gripbox

    if(DEBUG){printf("distance('%s', '%s') = %f\n", gripper->GetName().c_str(), prop->model->GetName().c_str(), distGoal);}

    // assign reward with the ratial of the distGoal using moving average distance.
    if( episodeFrames > 1 )
    {
        const float distDelta  = lastGoalDistance - distGoal;
        const float movingAvg  = 0.9f;

        // compute the smoothed moving average of the delta of the distance to the goal
        avgGoalDelta  = (avgGoalDelta * movingAvg) + (distDelta * (1.0f - movingAvg));
        rewardHistory = avgGoalDelta * REWARD_MULTIPLIER;
        newReward     = true;
    }

    lastGoalDistance = distGoal;
}
```

Fig. 6. distGoal based reward

When episode is time out, then give a negative reward to agent.

5 RESULTS

Below is the results of the DQL. The algorithm meets both objectives in a reasonable iteration. With the simulation,

```
// episode timeout
if( maxEpisodeLength > 0 && episodeFrames > maxEpisodeLength )
{
    // give timeout reward loss
    printf("ArmPlugin - triggering EOE, episode has exceeded %i frames\n", maxEpisodeLength);
    rewardHistory = REWARD_LOSS;
    newReward     = true;
    endEpisode    = true;
}
```

Fig. 7. Episoded timeout reward

it can be found that all the reward should be defined in appropriate way.Otherwise some penalty and some reward don't effect the total reward. With bad reward functions, manipulator can't learn how the action effects the results. If manipulator gets bad reward, then learning agent output a bad value for controlling the manipulator.

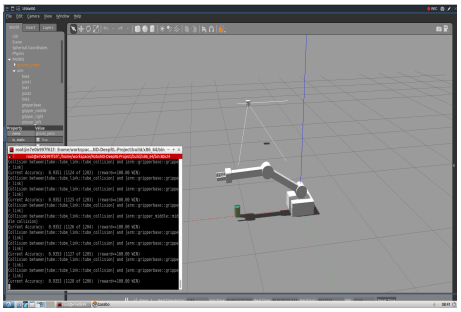

Fig. 8. Objective 1 with 93 percentage accuracy

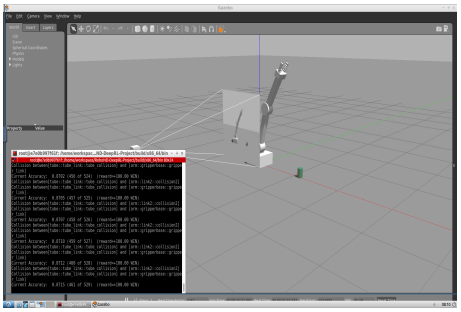

Fig. 9. Objective 2 with 87 percentage accuracy

6 FUTURE WORK

If the algorithm need better results, the reward functions should be defined more in detail. Parameters like roll, pitch , yaw of the gripper, gripper moving velocity, arm moving and so on should be defined and gives it a appropriate reward functions. There are still many things to be done for implementing the algorithm into real world. In the real word, items are not in a fixed position, gripper should grip a appropriate position and so on. DQL based algorithm should reward with a more carefully defined and more accurate reward functions.

7 REFERENCE

References:
1.www.udacity.com

2.https://medium.com/@fernandojaruchenunes/udacity-robotics-nd-project-8-deep-rl-arm-manipulation-ce480cbb1194

3.https://github.com/csosa27/RoboND-DeepRL-Project

4.http://gazebosim.org/tutorials