DeepRLArm Manipulation

Shweta Ruparel

1 Introduction

Deep Reinforcement Learning for Robotics is a paradigm shift. The basic idea is to start with raw sensor input, define a goal for the robot, and let it figure out, through trial and error, the best way to achieve the goal. In this paradigm, perception, internal state, planning, control, and even sensor and measurement uncertainty, are not explicitly defined. All of those traditional steps between observations (the sensory input) and actionable output are learned by a neural network.

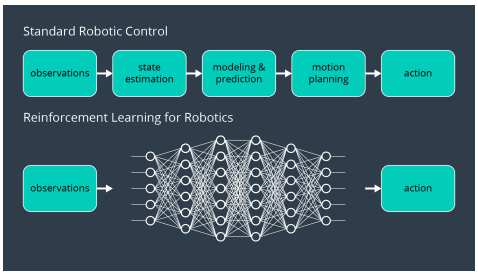

Fig. 1.

1.1 Reinforcement Learning

Let's revisit a few definitions and basics of Reinforcement Learning

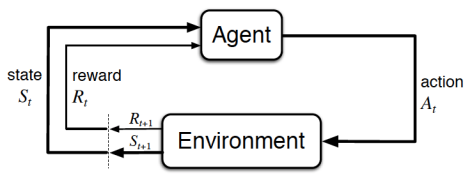

Fig. 2. The agent-environment interaction in reinforcement learning.

- The reinforcement learning (RL) framework is characterized by an agent learning to interact with its environment.
- At each time step, the agent receives the environment's state (the environment presents a situation to the agent), and the agent must choose an appropriate action in response. One time step later, the agent receives a reward (the environment indicates whether the agent has responded appropriately to the state) and a new state.
- All agents have the goal to maximize expected cumulative reward, or the expected sum of rewards attained over all time steps.

Episodic vs. Continuing Tasks

- A task is an instance of the reinforcement learning (RL) problem.
- Continuing tasks are tasks that continue forever, without end.
- Episodic tasks are tasks with a well-defined starting and ending point.
  - In this case, we refer to a complete sequence of interaction, from start to finish, as an episode.
  - Episodic tasks come to an end whenever the agent reaches a terminal state.

The Reward Hypothesis

Reward Hypothesis: All goals can be framed as the maximization of (expected) cumulative reward.

1.2 Summary

While reinforcement learning agents have achieved some successes in a variety of domains, their applicability has been limited to domains in which useful features can be handcrafted,or to domains with fully observed, low-dimensional state spaces.Recent advances in training deep neural networks to develop a novel artificial agent, termed a deep Q-network, that can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning.This work bridges the divide between high-dimensional sensory inputs and actions, resulting in the first artificial agent that is capable of learning to excel at a diverse array of challenging tasks.

2 Deep Reinforcement Learning

A key difference between RL and Deep RL is the use of a deep neural network. Think of the collection of value-action pairs that define what actions an agent should take in any situation as a function of the observations that the agent receives from its environment.

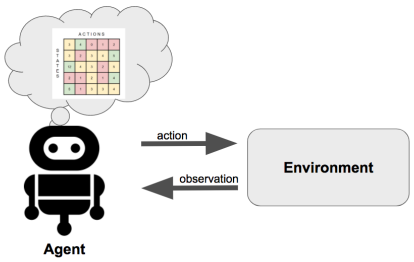

Fig. 3. The agent-environment interaction in reinforcement learning.

A neural network can be used to approximate this function because through its large quantity of parameters that can be “learned” through trial and error.

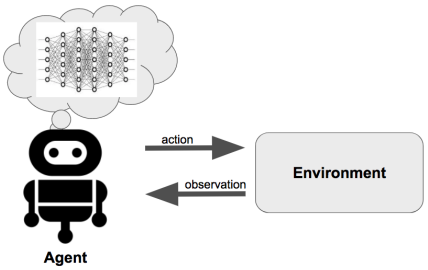

Fig. 4. The agent-environment interaction in deep reinforcement learning using neural network.

2.1 Test environment and Objective

This project is based on the Nvidia open source project ”jetson-reinforcement” developed by Dustin Franklin.The goal of this project is to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:

- Have any part of the robot arm touch the object of interest, with at least a 90 percent accuracy for a minimum of 100 runs.
- Have only the gripper base of the robot arm touch the object, with at least a 80 percent accuracy for a minimum of 100 runs.

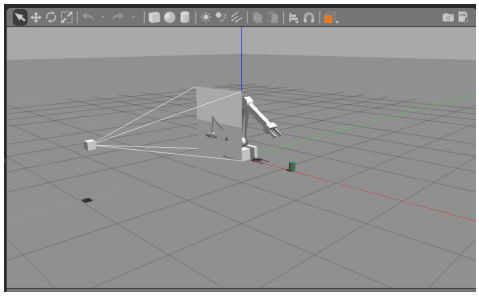

Fig. 5. Robotic Arm and a Prop

The project folder, RoboND-DeepRL-Project, is already loaded into the Udacity project workspace.Once the gazebo environment loads up, observe the robotic arm, a camera sensor, and an object in the environment. The following tasks are needed to be done and written in C++ to achieve the objective of this project.

- Subscribe to camera and collision topics.

- Create the DQN Agent. Refer figure below to check the DQN agent and camera and collision topics.

```
// CREATE DQN AGENT
if( !createAgent() ){
        printf("Error creatings DQN Agent\n");
        return;
}

// SUBSCRIBE TO CAMERA TOPIC

cameraSub = cameraNode->Subscribe("/gazebo/arm_world/camera/link/camera/image",
        &ArmPlugin::onCameraMsg, this);

// Create our node for collision detection

collisionNode->Init();

// SUBSCRIBE THE PROP COLLISION TOPIC

collisionSub = collisionNode->Subscribe("/gazebo/arm_world/tube/tube_link/my_contact",
        &ArmPlugin::onCollisionMsg, this);
```


Fig. 6. Create DQN and Subscribe to Camera Node and collision topics

- Velocity or position based control of arm joints.

```
#if VELOCITY_CONTROL
        if( velocity > VELOCITY_MAX )
                velocity = VELOCITY_MAX;
                vel[action/2] = velocity;
                for( uint32_t n=0; n < DOF; n++ )
                {
                        ref[n] += vel[n];

                        if( ref[n] < JOINT_MIN )
                        {
                                ref[n] = JOINT_MIN;
                                vel[n] = 0.0f;
                        }
                        else if( ref[n] > JOINT_MAX )
                        {
                                ref[n] = JOINT_MAX;
                                vel[n] = 0.0f;
                        }
                }
#else
// Increase or decrease the joint position based on whether the action is even or odd
// Set joint position based on whether action is even or odd.
        const int i = action / 2;
        const int c = 1 - 2 * (action % 2);
        float joint = ref[action/2] + c * actionJointDelta;
// limit the joint to the specified range
        if( joint < JOINT_MIN )
                joint = JOINT_MIN;
        if( joint > JOINT_MAX )
        joint = JOINT_MAX;

        ref[i] = joint;
#endif
return true;
```

Fig. 7. Position Based Control is used

- The next set of tasks are based on creating and assigning reward functions based on the required goals. There are a few important variables in relation to rewards
  - rewardHistory - Value of the previous reward, you can set this to either a positive or a negative value.
  - The values for positive or negative rewards, respectively.
  - newReward - If a reward has been issued or not.
  - endEpisode - If the episode is over or not.
- Reward for robot gripper hitting the ground
- Issue an interim reward based on the distance to the object. Refer the figure below that shows the rewards assigned for hitting the ground else assigning an interim reward based on the distance between the prop and the arm/gripper.

```
// Get the bounding box for the gripper
const math::Box& gripBBox = gripper->GetBoundingBox();

// Ground Threshold
const float groundContact = 0.05f;

// Set appropriate Reward for robot hitting the ground.
if( gripBBox.min.z <= groundContact || gripBBox.max.z <= groundContact )
{
        rewardHistory = 2*REWARD_LOSS;
        newReward     = true;
        endEpisode    = true;
}
// Issue an interim reward based on the distance to the object
else
{
// calculate the distance between two bounding boxes
        const float distGoal = BoxDistance(gripBBox, propBBox);
        if( episodeFrames > 1 )
        {
                const float distDelta  = lastGoalDistance - distGoal;
                avgGoalDelta           = (avgGoalDelta * alpha) + (distDelta * (1.0 - alpha));
                rewardHistory          = (avgGoalDelta) * REWARD_MULT;
                newReward              = true;
        }

        lastGoalDistance = distGoal;
}
```


Fig. 8. Reward for hitting Ground and Interim Rewards

- Issue a reward based on collision between the arm and the object.

```
Condition Check to see if any part of the arm collides with the prop-
First it checks if  Prop is hit-> it checks if it is  collision 2 OR if it is gripper link
  if (strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0  )
  {
  //Any part of the arm touching the prop will recieve the award to make it achieve 90% asap
        if(strcmp(contacts->contact(i).collision2().c_str(), COLLISION_ARM) == 0 || strcmp(contacts->
        contact(i).collision2().c_str(),COLLISION_POINT ) == 0)
        {
                if (DEBUG1) printf("I hit the prop, SUCCESS \n");
                        rewardHistory = REWARD_WIN;
                        endEpisode = true;
        }

        else
        {
                if (DEBUG1) printf("Missed the prop Reward %f \n", rewardHistory);
                rewardHistory = REWARD_LOSS ;
                        endEpisode = false;
                newReward  = true;
                        return;
        }
  }
```


Fig. 9. Reward for task1 any part of the arm hitting the prop

- Tuning the hyperparameters

```
HYPERPARAMTERS FOR TASK 1

#define INPUT_WIDTH         64              // Size of input will impact memory use
#define INPUT_HEIGHT        64
#define NUM_ACTIONS         DOF*2           // Each joint action is either decreas or increas. DOF is
number of joints
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.0001f
#define REPLAY_MEMORY       10000
#define BATCH_SIZE          128             // bigger size will require more computing power.
#define USE_LSTM            true
#define LSTM_SIZE           128
```


Fig. 10. HyperParameters tuned for task 1

- Issue a reward based on collision between the arm's gripper and the object.

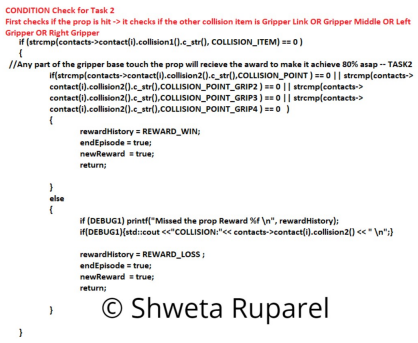

Fig. 11. Reward for task2 only gripper hitting the object

3 Reward Function

The rewards are created at different levels based on the task. **The rewards strategy for any part of the arm hitting the prop with 90 percent accuracy is stated below-**

- REWARD WIN is set to 0.20f
- REWARD LOSS is set to half of the winning award -0.10f
- To calculate the interim award the multiplier constant is set to REWARD MULT 20.0f
- To calculate the interim award that is smoothed moving average of the delta of the distance to the goal using the below formula- avgGoalDelta= (avgGoalDelta * alpha) + (distDelta * (1.0 - alpha));
  Here "alpha" is set to 0.7f , large value helped obtaining better accuracy.
- Twice the REWARD LOSS is assigned if the arm hit the ground to make sure higher penality is recieved to make a better choice next time.

The rewards strategy for gripper part hitting the prop with 80 percent accuracy is stated below-

- REWARD WIN is set to 0.40f (4 times REWARD LOSS).
- REWARD LOSS is set to -0.10f
- To calculate the interim award the multiplier constant is set to REWARD MULT 50.0f - more than the previous task , since the precision required is more and any right hit should get more reward.

4 HyperParameters

The following hyperparameters were tuned to get the accurate results.The hyperparameters are set using trial and error method

For task 1 - To achieve 90 percent accuracy for any part of the arm touches the prop.

- Learning Rate - The learning rate was experimented to change from 0.1.0.2,0.01,0.0001. The lower learning rate helped get better accuracy and the convergence to achieve 90 percent was achieved within 100 episodes.
- Input width and Input height is kept as 64 X64 as the bigger size can be very computation extensive.
- Optimizer was chosen to be "RMSprop". "Adams" can also be used to experiment
- BatchSize is kept at 128 and LSTM SIZE is kept 128 and "USE LSTM" is set to true.
- REPLAY MEMORY is set as default 10000

For task 2 - To achieve 80 percent accuracy for any part of the gripper touches the prop.This was more precise as only gripper hitting with the prop is considered.

All the hyperparameters except the batch size remained the same. Changing Batch Size from 128 to 32 clearly helped in obtaining accuracy much faster.

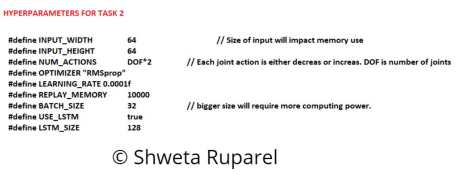

Fig. 12. HyperParameters tuned for Task 2

5 RESULTS / DISCUSSION

A DQN agent is created, reward functions are defined and Hyper Parameters are tuned for RL agent to train a robotic arm to achieve two objectives. Both the tasks were achieved successfully.

In task 1, the objective was that the any part of the arm touches the prop 90 percent of the time.This was achieved within 100 episodes.

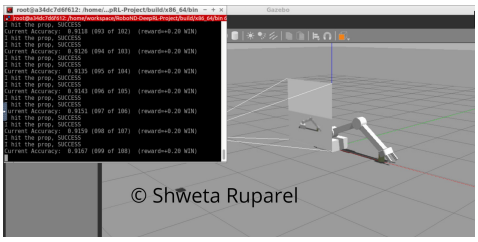

Fig. 13. 90 percent achieved for task 1 within 100 episodes.

In task 2, the objective was that the gripper touches the prop 80 percent of the time. This was achieved within 55 episodes.

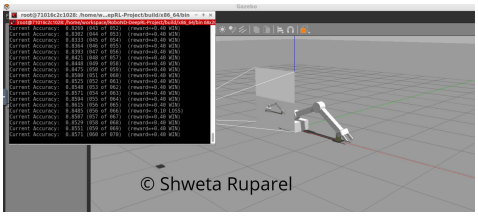

Fig. 14. 80 percent achieved for task 2 within 50 episodes

Overall the agent performs well for both the objectives.However, if the arm initially does not extend fully and hits the ground first just before the prop is located , it takes more time to learn to move towards the prop. If the arm extends and is almost on top of the prop it learns quickly since the interim award helps to determine if it is near the prop or far away.

6 FUTURE WORK

To improve the current results,a better reward functions needs to created. In the current reward function, until the very end of episode, the agent is blind as to where it is heading. At the very end, it may encounter a collision with a gripper(high reward) or a ground(high penalty).

If the arm is not extended full enough to be able to reach on top of prop and hits the ground much before in the front of prop . It takes more learning curve to reach upto the prop. But if the initial hit is almost in line to the distance where the prop is placed , it learns quickly and converges to hit at the right spot.

The randomness factor and the initial set of iterations can be experimented for better learning curves. Finding out what parameters would best ensure a better start to the learning, would be an interesting study.

Another interesting method would be to turn this task into a continuing task where the interim reward system would guide the arm to the prop in a few iterations.