Deep Reinforcement Learning

Snehal Gor

Abstract—Deep Reinforcement Learning (DeepRL) 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 (Function approximation). Author in this project applied Deep Q-Network (DQN) to train robotic a robotic arm. Author has successfully trained DQN in a Gazebo environment, whereby Arm has been fed captured images of the setup and reward as per the action. Author achieved more than 90% accuracy for objective 1, while near to 90% accuracy for objective 2. Further Author has also explored Q learning, PyTorch, Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) Network. Author here is documenting his learning, findings and results.

Index Terms—Robot, Udacity, LaTeX, DQN, RNN, LSTM, PyTorch, ROS, Gazebo.

1 Introduction

As age of data, artificial intelligence and robots dawn upon us, robots which can handle complex environment autonomously without any human interventions is becoming everyday reality. We see robots in every sphere of life, be it in industrial automation, entertainment, space exploration or Autonomous vehicle (and many more). Robots are making our world faster, safer, greener, smarter and enjoyable.

For robot to be autonomous in an environment without human intervention, exploration through trial and error to achieve a defined goal is a basic need. While Reinforcement learning being applied successfully to that goal, their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. Combining recent advances in Deep Neural Network (DNN) with reinforcement learning, Deep Reinforcement learning can learn successful policies directly from high-dimensional sensory inputs.

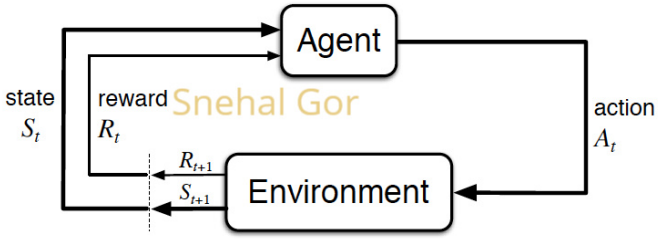

Fig. 1. The agent-environment interaction in reinforcement learning. [1]

Author has learned, understood and used Udacity shared PyTorch implementation of Deep Q-Network (DQN) [2]. In DQN reinforcement learning (Q learning algorithm) has been combined with convolutional neural network. CNN which uses hierarchical layers of tiled convolutional filters, to mimic the effects of receptive fields. Thereby exploiting the local spatial correlation present in images, and building in robustness against transformations.

In this report, Author is documenting his exploration of DeepRL using DQN in ROS and Gazebo environment. In subsequent section Author will document

- Background
- Project brief and Configuration
- Results
- Discussion
- Future Work

2 Background / Formulation

In Reinforcement learning, the robot interacts with an environment through a sequence of observations, actions and rewards. The goal of the robot is to select actions in a fashion that maximizes cumulative future reward.

A key difference between Reinforcement Learning and DeepRL is the use of a deep neural network (DNN or CNN). If we can think 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. Then a neural network can be used to approximate this function.

DQN is the first major architecture to deploy DNN for reinforcement learning problem. It's been extension from paper by Reed Miller and Martin on Neural fitted value iteration [3]. There few major ideas, which made DQN possible

- Replay Pool - Rolling history of past data (state, action and reward). By using replay pool and randomizing over data, DQN removes correlations in the observation sequence. Thereby making DeepRL learning stable.
- Target Network - Represents older Q-Network. Basically DQN maintains two networks, one for current learning updates and another to compute target prediction. By doing this we can decouple target / prediction from current learning / weight updates and make learning more stable.

In recent year, DeepRL has been further expanded with Recurrent Neural Network (RNN) and Long Short Term

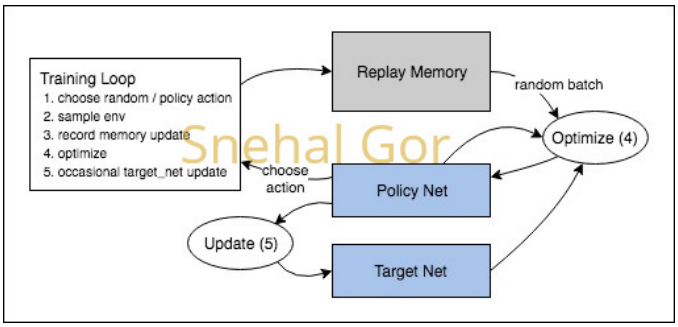

Fig. 2. The DQN implementation data flow. [4]

Memory (LSTM) architectures [5]. RNNs are a type of neural network which utilize memory, i.e. the previous state, to predict the current output. The shortcoming of RNNs are, they are unable to learn from long-term dependencies. This is handled by another architecture called Long Short Term Memory, or LSTM. The LSTM architecture keeps track of both, the long-term memory and the short-term memory, where the short-term memory is the output or the prediction. The architecture of LSTM consists of four gates that carry out specific functions - Forget, Learn, Remember and Use.

3 Project brief and Configuration

Author used Udacity shared framework, Gazebo and different ROS packages to train a DQN for robotic arm. The framework consists of a custom robotic arm, a RGB camera and a tube object in a Gazebo world and utilizes Pytorch DQN implementation and C/C++ APIs for DeepRL. Following is the brief explanation of the major files

- gazebo-arm.world - Describes the robotic arm setup in URDF format. It contains / calls gazebo plugin called the ArmPlugin. (libgazeboArmPlugin.so is attached to the robot model)
- ArmPlugin - This plugin is responsible for creating the DQN agent and training it to learn to touch the tube object in gazebo world.
- ArmPlugin.cpp - The plugin is defined in this file. Few important functions are described below
  - Load: Creates and internalizes nodes that subscribes to camera and contact sensor topics for the tube object.
  - onCameraMsg: Callback function for the camera subscriber. It takes camera topic message, saves the image and pass it further to DQN.
  - onCollisionMsg: This function is used for catching collision events with tube object (specifically with my_contact sensor [6])
  - createAgent: Creates and initializes the DQN agent.
  - updateAgent: Receives the action value from DQN, and decides to take action - controlling the robotic arm joints. There are two possible ways to control arm joints.
    - Velocity control : It takes into account the joints velocity for better control / realistic motion [7].
    - Position control : It uses simpler joint position based control. Which sometimes create not so smooth / realistic motion.
  - OnUpdate: Primarily utilised for issuing different rewards and train DQN.

Below Table 1 provides brief the DQN setups for

- Objective 1: Have any part of the robot arm touch the object of interest.
- Objective 2: Have only the gripper base of the robot arm touch the object.

TABLE 1
Brief details of Author's DQN configuration

	Objective 1	Objective 2
Joint Controls	Position	Position
Reward for Hitting Ground	-1.0	-1.0
Interim Reward for avg_delta ¿ 0	0.0	0.0
Interim Reward for avg_delta ¡= 0	-0.1	-0.1
Collision Reward	1.0	1.0
Collision Objects	Gripper Base, Middle, Left and Right	Gripper Base
End of Episode Reward	-1.0	-1.0

Further Author kept different hyper-parameters as per below table 2

TABLE 2
Brief details of Author's DQN Hyper-parameters tuning

	Objective 1 and Objective 2
Image W X H	64 X 64
Optimizer	RMSProp
Learning Rate	0.01
Batch Size	32
Use LSTM	false

3.1 Author's Reasoning for Selection of different Rewards and Hyper Parameters

Below Author is sharing his reasoning for selection of different values

- Interim Rewards - Author wanted arm to move towards the tube object, while wanted to penalize movement away from tube object. So Author kept -0.1 reward when average delta (Calculated as average between current and previous average) stayed negative or zero. Reward has been kept to 0, for average delta value greater than 0.
- Negative Rewards (-1.0) has been kept for end of episode and gripper hitting the ground plane.
- Positive Reward of 1.0 provided for every successful collision of Gripper assembly for Objective 1, while only Gripper base for Objective 2.
- Author used Position control for joint movements, as it was simpler and easier to control. With velocity

control Author observed better arm movement, but found it difficult to control.
- Author reduced Image width and height to 64 X 64, as keeping value at 256 X 256 and above framework threw an error.
- Author use RMSProp as optimizer with learning rate at 0.01 and batch size at 32
- Author didn't use LSTM, as DQN performed quite well. Author do feel that LSTM may provide better performance than DQN, due to leveraging state history better.

4 Results

Author got a good result. With Objective 1 Authour achieved near to 92% accuracy, while for Objective 2 Author got near to 90% Accuracy. Snapshots of the both environments have been provided below

- Snapshot of Accuracy and Robot touching the Tube object for Objective 1 - Fig 3
- Snapshot of Accuracy and Robot touching the Tube object for Objective 2 - Fig 4

Learning while for the project, further shared in the Discussion section 5.

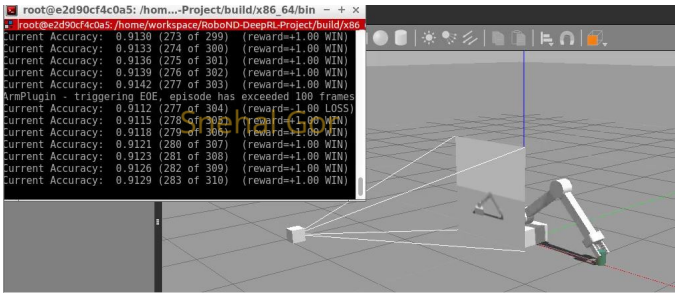

Fig. 3. Objective 1 Output Snapshot.

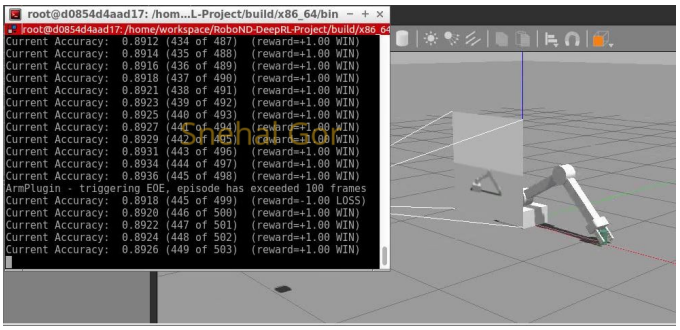

Fig. 4. Objective 2 Output Snapshot.

5 Discussion

Author could successfully achieve both objectives. In Author's opinion following are few of the learning / reasons of getting a good performance

- Simplicity of the environment : Since arm motion has been restricted to one plane, problem become simpler.
- Designing of Rewards was optimal, which helped Arm learn quickly. Also precision of arm while colliding with tube object is excellent.
- DQN is one of the impressive approach to DeepRL, which combines DNN / CNN and reinforcement learning.

Author faced some challenges. Below Author is sharing learning

- Code compilation error due to missing dependency of *libignition-math2-dev*. Author installed the dependency using *sudo apt-get install*
- Collision identification and implementation. Author is sharing few of the references, which helped in understanding it better [6].

6 Conclusion / Future work

Author has successfully explored Reinforcement learning, DeepRL, Deep Q-Network, RNN and LSTM. Further Author successfully trained robotic arm in Gazebo environment using C++ APIs. Author would like to further explore velocity control mechanism and LSTM in further details. Author sees DeepRL and related approaches, having numerous application from Autonomous car, factory automation, space exploration, entertainment, etc.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*. Adaptive computation and machine learning, MIT Press, 1998.
[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[3] M. Riedmiller, "Neural fitted q iteration first experiences with a data efficient neural reinforcement learning method," in *In 16th European Conference on Machine Learning*, pp. 317–328, Springer, 2005.
[4] *Reinforcement Learning (DQN) Tutorial*.
[5] *Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM)*.
[6] *Topics Subscription*.
[7] *Mobile Robot Dynamics*.