Deep Reinforcement Learning Manipulator

Ross Lloyd

Abstract—A DQN Agent is applied to a virtual 3 DOF robotic manipulator arm. A set of hyperparameters for the DQN Agent and appropriate reward functions and values were determined that allowed the agent sufficient control accuracy to touch a simulated object either with any part of the manipulator arm, or with a specific part of the robot arm, namely the gripper base. Parameters were successfully derived and the agent carried out the tasks adequately.

Index Terms—Pytorch, DQN, DNN, Gazebo, Reinforcement Learning, Manipulator.

1 Introduction

Deep Reinforcement Learning, a fusion of Q Learning and Deep Learning approaches, is an extremely powerful branch of AI that can allow an "agent" to learn quickly and efficiently to learn how to perform a task, given only the world state, a set of actions and some reward associated with those actions and states. In this project, a Pytorch c++ API based DQN agent will interface with a 3 dof robot arm in gazebo and learn to contact an object of interest.

2 Background

In order to carry out the task, a virtual arm and a target object (a small cylinder with associated gazebo contact sensor object) were built in the gazebo simulator. The arm is a simple 3 degree of freedom (DOF) manipulator, though for the purposes of the assigned tasks, one DOF - the base - was constrained. The controls for the joints are exposed by a gazebo plugin (armplugin.h). This object also contains member variables for a number of relevant arm parameters such as joint angles, collision detection, velocity, variables relating to the DQN RL agent such as rewards, and distance to goal. The DQN agent has access to to the plugin via the gazebo plugin shared object file, libgazeboArmPlugin.so, attached to the robot model in gazebo-arm.world. The DQN agent also has access to a virtual camera, whose size is controlled by the DQN hyperparameters. At each timestep, subscriber functions use callbacks to update the agent based on whether a new camera frame is received or a collision is detected, another function allows the DQN to take and issue actions / commands, and another issues rewards based on the outcome. See submitted code submission for details. Two options for joint control were presented, and although velocity control was successfully used for the gripper-only scenario, the position only control method was found to be more robust over both scenarios.

3 Methodology

3.1 Parameters

The tables below list the selected parameters for this project with an overview of why they were selected.

3.1.1 *Epsilon-Greedy and DNN Hyperparameters*

Parameter	Value	Justification
INPUT CHANNELS	3	RGB Input Channels
ALLOW RANDOM	TRUE	Enables epsilon-greedy algorithm to take random actions
GAMMA	0.9	Set discount rate to 0.9. This is a common value that gives effective results, favouring earlier rewards over later ones. This encourages the agent to find a solution more quickly
EPS START	0.9	Set the initial probability of selecting a random action to 0.9. This value ensures that the arm will make sufficient random moves to learn optimal action value pairs
EPS END	0.02	Set the final probability of selecting a random action to 0.02. This is set lower than default as the arm continued to select random movements, sometimes leading a series of successes to fail
EPS DECAY	200	Default value for the epsilon greedy decay rate. This parameter controls how quickly the robot stops random exploration and moves to 'exploit' known good action-value pairs.

The Epsilon-Greedy algorithm ensures a trade off between the amount of time the robot spends 'testing' the results of random actions, and exploiting those it has already found to be effective. The aim is to have the arm initially explore a lot, given by a high value for the the EPS START parameter, but gradually decrease as time goes on, given by the much lower EPS END parameter. The rate at which the robot reduces its exploring is controlled by the decay parameter. A poor balance in these values can lead to an agent that fails to find profitable actions and so never reaches the goal, taking actions that only accomplish part of the goal or receive sub-optimal rewards. Conversely it may persist in taking random actions that can prevent it from ever capitalising on the knowledge it has gained.

3.1.2 Deep Neural Network Hyper-Parameters

Parameter	Value	Justification
INPUT WIDTH	128	Sets the width of the input area for the camera. Setting to 64 caused the arm not to learn from rewards when received, setting higher lead to poor performance. 128 was a good trade off.
INPUT HEIGHT	128	See above for height
OPTIMIZER	RMS Prop	Default optimizer. Another option would have been Adam, however this one performed adequately
LEARNING RATE	0.1	A commonly recommended value for learning rate
REPLAY MEMORY	10000	This is the default parameter
BATCH SIZE	256	Controls the number of training examples used per iteration. This value was a good trade off between having the arm learn from rewards and performance. Set lower than this and the arm would continue iterating without learning
LSTM SIZE	256	This was a commonly used parameter in the literature. Setting it to higher or lower values did not appear to have as much of an impact on results as batch size and so it was left at this value.

The **replay memory** parameter controls how many observed transitions are stored. Through random sampling, the transitions building a batch are decorrelated. The benefit is stabilization and improvement of the DQN training procedure.

Learning Rate: The process used to train Deep learning neural networks is known as the stochastic gradient descent algorithm. This algorithm is responsible for estimating the error gradient for the current model state, using examples drawn from the training dataset. Using back propagation it then updates the weights of the model. The learning rate controls the amount that the weights are updated during training. Setting this value appropriately leads to an agent that will converge in a timely manner without experiencing overfitting.

LSTM: The two parameters, one to activate it, and one to define the size, help the agent to learn from episodes that happened in the past and recently, and allows them to make use of long term dependencies.

The above hyperparameters were used for both tasks.

3.1.3 Reward Parameters

Parameter	Value	Justification
REWARD WIN	50.0	A higher value was selected for the reward to maximise the impact of a correct outcome. When set to the same as the loss function (but positive), the agent would not seem to learn as well from good outcomes.
REWARD LOSS	-20.0	This value was effective in discouraging the arm from taking incorrect actions, but not so much that the arm was unable to move.
REWARD BOOST	4.0	This value aided in increasing the interim reward received for movement towards the goal. Too high and the arm would tend to overshoot the object, too low and it would not approach the object enough.
ALPHA	0.4	Again, this value was found as a compromise of allowing the arm to approach the robot without overshooting.

To assess collisions in cases other than floor contact, the member variable *contacts-¿contact(i).collision2()* was compared with strings representing the desired arm model components. This variable is associated with a gazebo contact sensor object.

Positive Rewards were assigned when any part of the arm touched the object in the case of the first objective, and when only the gripper base touched the object as in the case of the second objective. An End Of Episode (EOE) event was also sent in the case that the correct part of the arm contacted the object. A small interim reward, based on the distance to the goal, was also assigned for every step closer the target area (the middle of the gripper) moved closer to the object. This value was multiplied by a booster value to increase its effect. Naturally no EOE event was sent after issuing the interim reward. The effect of this interim reward was to motivate the gripper towards the goal at each time step.

Negative Rewards were assigned to the *robot "hitting" the ground*, calculated as the minimum limit of a gazebo bounding box around the middle of the gripper coming within a threshold value of the ground plane. This also caused an End Of Episode (EOE) event to be sent to the agent. A negative reward and EOE were also issued if *100 frames passed* without an EOE event or the arm touching the object as desired. In the case of the gripper-contact-only requirement, a negative reward and EOE event were also issued if any part of the arm other than the gripper base contacted the object. Note that a small negative reward was added to the interim reward at each time step. As the reward for movement towards the goal could also encourage the gripper to overshoot, this small negative reward helped restrain the arm's movement and resist overshooting in the case of the gripper-only contact. For the gripper-only contact this value was set to 0.35 to aid in guidance of the smaller gripper base towards the goal, and for the arm-only contact it was 0.3.

Alpha: The weighting parameter for the smoothed moving average. The current smoothed value is a weighted average of the current value and the previous smoothed value. If alpha is equal to 0, then the current smoothed point is equal to the previous smoothed value. If alpha is equal to 1, the current smoothed point is set to the current value. So the closer alpha is to 1, the less the prior data points will affect the outcome of the smoothing function.

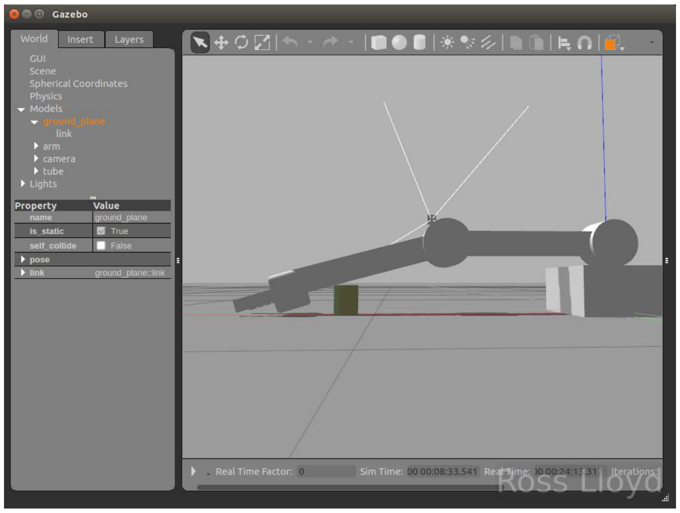
The above reward parameters were used for both tasks.

4 Results

The robot successfully touched the arm in the two required ways, firstly with any part of the robot arm, and then with only the base of the gripper. As can be seen from figures 3 and 4, in both cases the required accuracies were achieved over 100 runs. There was a fair amount of randomness to how well this was accomplished especially for the gripper-base only, with the robot sometimes converging on the best solution very quickly (1-50 runs) and other times taking much longer (100-200 runs). This mainly seemed to be linked with how early on the arm took the random movements that lead to the optimal arm position which was the elbow-up position. Sometimes it would arrive at this very quickly, and other times it would overshoot the object and hesitate, then change the angle of joint 1 to bring the gripper inward. It would follow this with a downward movement of joint 2 to touch the object. Once this had been repeated 2-3 times, the arm could then repeatably reach its goal.

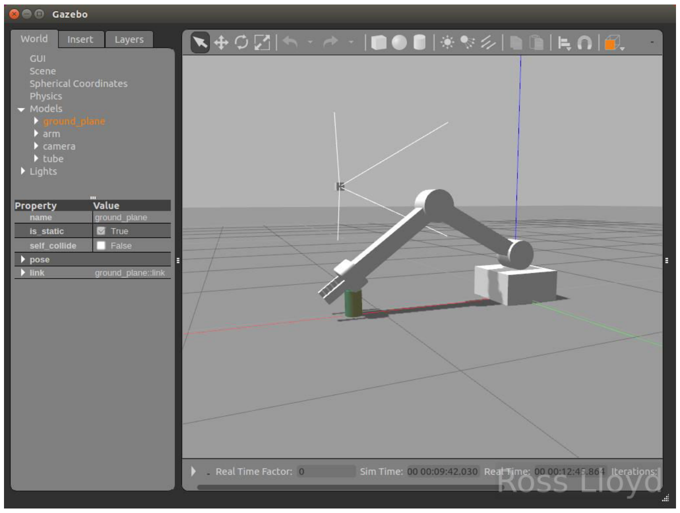
Figure 1: The requirement to have any part of the arm touch the object. Here we can see link2 in contact. In order to achieve this repeatedly it was necessary to set the increment angle to 0.1 per step as this increased the likelihood of the arm contacting the object without also striking the floor. The small reward penalty at each time step could also be relaxed to 0.3 as it was not so critical to prevent the gripper overshooting.

Figure 2: Gripper base only touching the object. Touching any other part of the gripper or arm lead to a EOE and negative reward being assigned. An increment angle of 0.15 was helpful in leading the arm to assume the most useful combination of angles through random actions earlier on. Before reaching the correct solution, the arm could either overshoot and contact with an undesired part of the arm, or alternately seem to get stuck exploring the area immediately in front of



Property	Value
name	ground_plane
is_static	True
self_collide	False
pose	
link	ground_plane::link

Fig. 1. Arm touching target object.



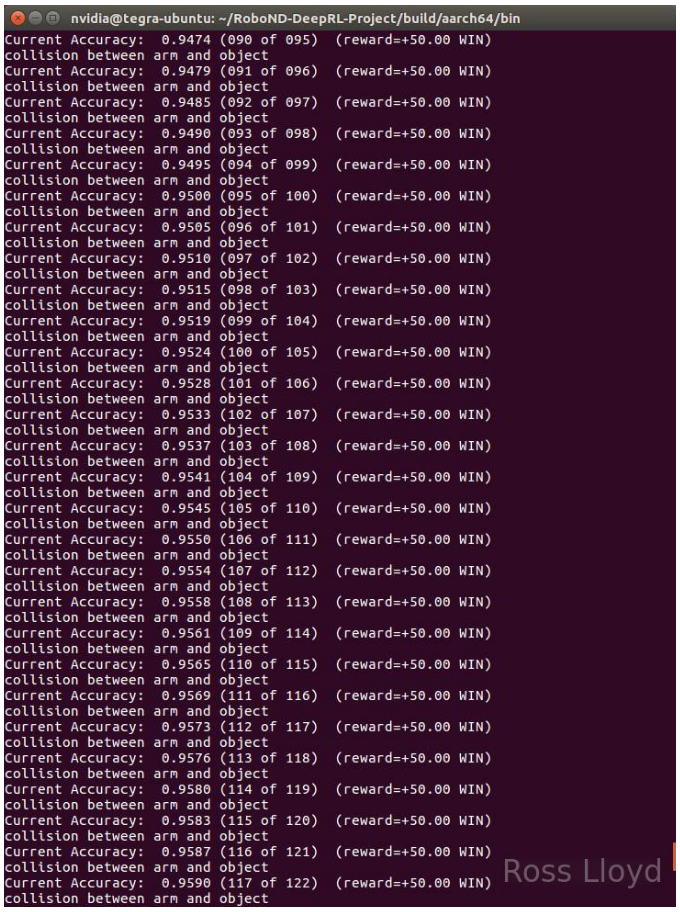
Property	Value
name	ground_plane
is_static	True
self_collide	False
pose	
link	ground_plane::link

Fig. 2. Arm touching object only with gripper.

the object. Due to the threshold setting for ground contact, it could occasionally be observed that ground contact would not be detected. However increasing this parameter could also lead to spurious contacts occurring when in fact the arm had contacted the object, due to the way the gazebo model responds to impacts.

5 Discussion and Conclusion

The use of Deep RL in the task of training a robot arm to make contact with a simulated object was shown to be feasible and reasonably robust. It is highly reliant on the outcome of the epsilon-greedy algorithm and which random actions it takes, with the agent tending to find the correct solution much more quickly if the correct combination of angles is hit upon early. If not, the robot can become more hesitant to take even correct actions as negative rewards accumulate. Additionally, as the accuracy is an average of all results so far, the gain in accuracy at each time step becomes steadily smaller. This means that if the arm hits upon the correct solution later in the training process, it can take much longer to finally reach the desired accuracy.


nvidia@tegra-ubuntu: ~/RoboND-DeepRL-Project/build/aarch64/bin
Ross Lloyd

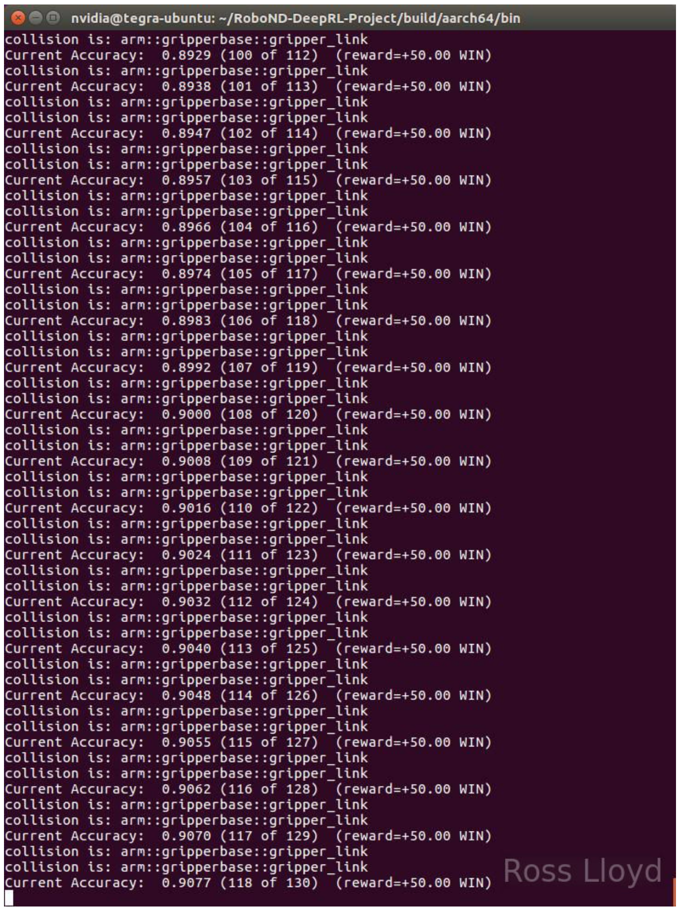
```
Current Accuracy:  0.9474 (090 of 095)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9479 (091 of 096)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9485 (092 of 097)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9490 (093 of 098)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9495 (094 of 099)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9500 (095 of 100)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9505 (096 of 101)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9510 (097 of 102)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9515 (098 of 103)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9519 (099 of 104)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9524 (100 of 105)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9528 (101 of 106)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9533 (102 of 107)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9537 (103 of 108)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9541 (104 of 109)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9545 (105 of 110)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9550 (106 of 111)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9554 (107 of 112)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9558 (108 of 113)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9561 (109 of 114)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9565 (110 of 115)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9569 (111 of 116)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9573 (112 of 117)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9576 (113 of 118)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9580 (114 of 119)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9583 (115 of 120)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9587 (116 of 121)  (reward=+50.00 WIN)
collision between arm and object
Current Accuracy:  0.9590 (117 of 122)  (reward=+50.00 WIN)
collision between arm and object
```

Fig. 3. Terminal output showing accuracy of 95 percent over 100 runs for contact of any arm component.

5.0.1 Improvements and further work

A more robust collision detection approach would have aided with accuracy, particularly in detecting ground collisions. One improvement that may be applied to the results here is the use of Actor-Critic RL algorithms, as these have been shown to converge more rapidly and accurately on correct solutions. As this project was carried out on the TX2, improvements in processing speed and possible improvements to hyperparameters could be realised by using more powerful GPUs to carry out the computations. Whilst position control was used here, in reality a manipulator arm would be controlled by velocity commands.

There is much real world work being done on manipulators and Deep RL and this is an extremely useful application. Allowing a manipulator to reach its goal without having to construct elaborate mathematical models is a significant time and cost saving measure anywhere manipulators are required to locate and accurately pick and place objects.


nvidia@tegra-ubuntu: ~/RoboND-DeepRL-Project/build/aarch64/bin
Ross Lloyd

```
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8929 (100 of 112)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8938 (101 of 113)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8947 (102 of 114)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8957 (103 of 115)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8966 (104 of 116)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8974 (105 of 117)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8983 (106 of 118)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.8992 (107 of 119)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9000 (108 of 120)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9008 (109 of 121)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9016 (110 of 122)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9024 (111 of 123)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9032 (112 of 124)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9040 (113 of 125)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9048 (114 of 126)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9055 (115 of 127)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9062 (116 of 128)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9070 (117 of 129)  (reward=+50.00 WIN)
collision is: arm::gripperbase::gripper_link
collision is: arm::gripperbase::gripper_link
Current Accuracy:  0.9077 (118 of 130)  (reward=+50.00 WIN)
```

Fig. 4. Terminal showing accuracy over 90 percent for gripper-only contact.

6 REFERENCES

- Udacity Robotics Nanodegree Class Materials
- Reinforcement Learning - An Introduction: *Sutton and Barto*
- https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/exposmoo.htm