Deep Reinforcement Learning for Guiding a Robotic Manipulator

Christopher T. Angell

Abstract—Robots typically have to be pre-programmed with specific movements and work in a highly constrained environment to accomplish their tasks. But what if the robot could learn to execute tasks in a dynamic way? Deep reinforcement learning aims to accomplish this by combining existing experience with Q-learning, a type of reinforcement learning where to value estimation of actions are updated based on perceived awards and values of adjoining states, with deep neural networks to provide a natural way to move from visual data to learned action. Here, we train an arm to reach for a target on the ground using deep reinforcement learning, demonstrating the principles at work.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Reinforcement learning, Deep Q-learning.

✦

1 INTRODUCTION

Robots traditionally have excelled at repetitive tasks in highly controlled environments. They are manually programmed for the tasks, and the work flow is known precisely. When robots are moved from such an environment to more complex environments and tasks encountered in the real world, it can be challenging to program them in a dynamic way to handle all cases. Robots that can learn from new experience and modify their behavior to suit the situation will allow robots to move in to a broader application domain. Reinforcement learning is the process whereby an agent, such as a robot, learns to act by receiving incremental rewards for actions. Q learning is one such approach in the domain of reinforcement learning where action rewards are mapped to states assuming that the agent can be modeled by a Markov decision process where future rewards depend only on the current state, and not on the history of states occupied. Traditional Q-learning suffers in high dimensional state spaces where every state needs to be learned separately, resulting in frequently using approximation functions to simplify the state space with hand-coded features. To circumvent the need for hand-coding, deep Q-learning uses a deep neural network as the approximation function allowing to map directly from a high dimensional input state space, such as visual data, to an action space [1].

Here we apply Deep Q-learning to the problem of a robotic manipulator reaching for an object (see Fig. 1). The arm position is recorded by a camera, and is fed into a neural network, with a reward function provided both by measuring the distance from the arm to the object, as well as recording collision with the target. Our network learns to reach the target first by rewarding a collision with any part of the robot arm, and then by measuring a reward only by colliding with the base of the gripper. We first give a brief background on reinforcement learning, then introduce our reward functions, explain our hyperparameters, and then introduce our results.

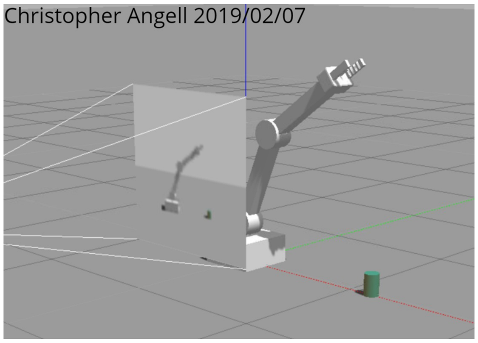

Fig. 1.

2 BACKGROUND

Reinforcement learning is related to the problem of optimal control [2]. Because of the highly-dimensional state space, it also suffers from the curse of dimensionality, which is that as the parameter space increases exponentially with the number of additional parameters as combination of parameters have to be taken into account. Learning in robotics can be split into three categories: model learning, reinforcement learning, and learning from expert demonstration. For reinforcement learning, there are two major areas: Value function methods and policy search. Value function methods can be further split into two areas: rollout based Monte Carlo methods and temporal difference methods, e.g. Q learning. Policy search methods can also be split into two approaches: gradient based approaches and likelihood ratio methods. The difference between methods is that value function methods suffer more strongly from the curse of dimensionality, while being able to explore more of the function space and find a global optimal. But policy search methods are more resilient in the face of highly dimensional search spaces, but are more likely to provide only a locally optimal solution.

Deep Q-learning approaches the problem of curse of dimensionality by replacing the highly dimensional state space, which encodes every single state combination possible in the system, with a neural network that takes limited state information and turns that into action space. So rather than needing to learn the Q-value for every state-action pair, a network policy is learned that maps state information to action. The deep aspect of this is that rather than processing sensory information through a pipeline to get discrete state information, the sensory information itself can be fed into the deep neural network, thus the network learns through reinforcement the best intermediate representation for states, and best learns how to map those states to actions.

To make Deep Q-learning work requires two further extensions. First, experience replay was used which is that experiences are stored as a set of state, action, reward values, which rather than training the network always from the latest experience, a set of experiences are stored and learned from in batches. This avoids the network from being subject to over training due to repetitive similar episodes. Second, fixed Q targets were used. This avoids the problem of having the Q targets being updated during learning which causes highly unstable training of the network. So two networks were used, the one where the action weights were updated, and a second where the Q-values were kept fixed and updated only at fixed intervals.

3 REWARD FUNCTIONS

Reinforcement learning hinges on specifying appropriate rewards for actions. Here we have created rewards for specific behavior in an effort to teach the manipulator arm to reach for an object.

- Distance to object – The primary award is based on change of distance between the manipulator end and the object of interest. The change of distance was smoothed over neighboring values to reduce noise:

  $$|\Delta_d| = \alpha|\Delta_d| + (1-\alpha)\Delta_d$$

  where $|\Delta_d|$ is the average (smoothed) change distance, and Δ_d is the change in distance from the gripper to the object of interest between the last frame and the current frame. α is the constant of smoothing. A reward of $|\Delta_d|$ was rewarded each time step.
- Ground collision – a reward of -1.0 was rewarded if it was detected that the gripper collided with the ground. This was measured based on the bounding box of the gripper having a z value less than 0.
- Exceeding time limit – If the time limit (set at 100 frames) of the collision was exceeded then a reward of -1.0 was rewarded, and the session was restarted.
- Collide with item – If the arm collided with the item, then a reward of +1.0 was awarded. Two different collisions were detected. First, a collision with the whole arm was trained against. Second, a collision only with the gripper base and the object was trained against.

4 HYPERPARAMETERS

The reinforcement learning for the manipulator arm depended on a number of hyperparameters. These parameters were adjusted to improve learning rate.

Christopher Angell 2019/02/07

```
Current Accuracy:  0.9873 (078 of 079; 341 of 379)  (reward=-1.00 LOSS)
Current Accuracy:  0.9875 (079 of 080; 342 of 380)  (reward=+1.00 WIN)
Current Accuracy:  0.9877 (080 of 081; 343 of 381)  (reward=+1.00 WIN)
Current Accuracy:  0.9878 (081 of 082; 344 of 382)  (reward=+1.00 WIN)
Current Accuracy:  0.9880 (082 of 083; 345 of 383)  (reward=+1.00 WIN)
Current Accuracy:  0.9762 (082 of 084; 345 of 384)  (reward=-1.00 LOSS)
Current Accuracy:  0.9765 (083 of 085; 346 of 385)  (reward=+1.00 WIN)
Current Accuracy:  0.9767 (084 of 086; 347 of 386)  (reward=+1.00 WIN)
Current Accuracy:  0.9770 (085 of 087; 348 of 387)  (reward=+1.00 WIN)
Current Accuracy:  0.9773 (086 of 088; 349 of 388)  (reward=+1.00 WIN)
Current Accuracy:  0.9775 (087 of 089; 350 of 389)  (reward=+1.00 WIN)
Current Accuracy:  0.9778 (088 of 090; 351 of 390)  (reward=+1.00 WIN)
Current Accuracy:  0.9780 (089 of 091; 352 of 391)  (reward=+1.00 WIN)
Current Accuracy:  0.9783 (090 of 092; 353 of 392)  (reward=+1.00 WIN)
Current Accuracy:  0.9785 (091 of 093; 354 of 393)  (reward=+1.00 WIN)
Current Accuracy:  0.9787 (092 of 094; 355 of 394)  (reward=+1.00 WIN)
Current Accuracy:  0.9789 (093 of 095; 356 of 395)  (reward=+1.00 WIN)
Current Accuracy:  0.9792 (094 of 096; 357 of 396)  (reward=+1.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.9691 (094 of 097; 357 of 397)  (reward=-1.00 LOSS)
Current Accuracy:  0.9694 (095 of 098; 358 of 398)  (reward=+1.00 WIN)
Current Accuracy:  0.9697 (096 of 099; 359 of 399)  (reward=+1.00 WIN)
Current Accuracy:  0.9700 (097 of 100; 360 of 400)  (reward=+1.00 WIN)
```

Fig. 2.

- $alpha$ – the derivative smoothing parameter. It was initially set to 0.5, but was reduced to 0.2 and finally to 0. It was found that lower amounts of smoothing improved learning drastically.
- Learning rate. The learning rate was adjusted from 0.1 to 0.3, and little dependence was found on it for the actual learning of the system. Because of this, a value of 0.2 was used.
- Optimizer. The optimizer used was the RMSprop optimizer for gradient descent.
- Replay memory. The number of replay sequences kept for experience replay. A replay memory of 10000 was initially used, but this was found to cause memory issues. It was successively lowered until a value of 1000 was used as it prevented memory errors from occurring.
- Batch size. A value of 8 was originally used, but was lowered to 4 as part of the effort to prevent memory errors.
- Arm speed. A higher arm speed of 0.15 radians per frame was initially used, but the arm would collide with the object of interest and then the ground in rapid succession such that it would be measured as a loss, so this was cut in half so that the arm would more slowly approach the object such that every collision would be properly registered.

5 RESULTS

For the task of any part of the arm touching the object as generating a reward, learning was quite rapid, and after 400 trials had an accuracy of 97% based on the previous 100 trials (see Fig. 2). For the task of only the gripper base touching the object generating a reward, the network had a much more difficult time of training, and only managed to reach 85% accuracy on previous 100 trails after training for 500 trials (see Fig. 3). A video recording the learning process is available online [1].

6 CONCLUSION / FUTURE WORK

In conclusion, we successfully implemented a Deep Q network to train a robotic manipulator arm how to reach an

---

[1] https://github.com/ctangell/RoboND-DeepRL-Project/blob/master/DeepRL.mp4

Christopher Angell 2019/02/07

```
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Current Accuracy:  0.8586 (085 of 099; 386 of 499)  (reward=-1.00 LOSS)
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Current Accuracy:  0.8500 (085 of 100; 386 of 500)  (reward=-1.00 LOSS)
Collision between[tube::tube_link::tube_collision] and [arm::link2::collision2]
Collision between[tube::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
```

Fig. 3.

object. Here, we trained only for a limited number of trials. This work could be improved upon by allowing the network to train for more trials. We could also further refine the reward function to use more of the visual information such as robot pose, but this could fall in the category of hand-coding features which in some ways defeats the advantage of using a Deep Q-network for reinforcement learning. Further more, this work did not include any of the more recent advances in reinforcement learning, such as actor-critic networks, or dueling Q-networks, which could improve our ability to train the networks more rapidly and reliably for this task. This work could also be expanded by considering more difficult scenarios such as teaching the network to close the gripper on the object, pick it up, and move it to a new location. Such sequence of actions may be more difficult to train, and may need to use demonstration learning methods to accomplish. It would be interesting to see how well the current algorithm performs at such tasks, and how we could expand it.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[2] J. Peters *et al.*, "Robot learning," in *Handbook of Robotics*, ch. 15, pp. 357–394, Berlin: Springer-Verlag, 2016.