RoboND-DeepRL-Project

Kevin James

22 October 2018

Introduction

The project uses Reinforcement Learning to train a robot arm to touch a target object. Specifically, a deep learning agent is used to learn the Q function relating state and action to reward. The agent receives the difference in camera images from one state to the next, and learns how to maximize overall reward from the actions it generates. The framework used is based on the Nvidia jetson-reinforcement project (https://github.com/dusty-nv).

Configuration

As shown in the images below, a *gazebo* environment is configured with a robot arm, target cylinder, and camera. The arm features two moving joints allowing it to extend along the x axis. The goal is to touch the cylinder without contacting the ground. In the first task it suffices for the arm to touch the cylinder, while in the more complicated second task, the the arm's *gripperbase* link must touch the cylinder. The problem is constructed such that contact between other parts of the arm results in failure.

End of Episode

Training proceeds over episodes of up to 200 frames. Each episode ends either when the arm makes contact with the ground or the cylinder, or when the frame count is exceeded. At this point a final reward is issued.

Reward Function

The agent receives a reward of +100 if it makes the required contact. Conversely, the agent receives a reward of -100 if the arm contacts the ground or exceeds 200 frames run length. A negative final reward is also received for failing to *win* but the amount of the reward depends on

the circumstances of the failure: -50 for contacting the cylinder with the *gripper_middle* link, and -25 for contacting the cylinder with other parts of the arm. Variation in the final reward for failure is intended to help the agent learn to approach the intended goal with greater fidelity. In earlier training with a fixed -100 reward for any failure, observation on the second task suggested that the information conveyed did not easily lead to convergence on the subtle detail of exactly where to make contact. Instead, nearly identical circumstances could result in +100 or -100. (The idea of varying loss was raised on *slack* and taken to be acceptable.)

To allow the agent to learn how to approach the goal, an interim reward is also generated. This reward is a function of the distance between the arm's contact surface and the cylinder. There are three components to the reward: distance based, correctness of change in distance, and a penalty for each step:

reward = (2.5 - squared_distance)/2.5 - (distance_increased ? 1.0 : 0.0) - 1.0

The reward was determined experimentally, and is intended to decline smoothly as distance to the goal is reduced while penalizing moving away from the goal. Rewards that eliminated the time penalty or otherwise generated positive net interim reward did not perform well. This may be due to the agent's ability to hover in place collecting positive reward instead of pushing toward the goal. The explicit gradient direction penalty seemed to be helpful in guiding actions toward the goal.

Hyperparameters

The network is augmented with a *replay memory* and is configured with multiple settings as shown in the table below. While optimizer, replay memory size, and batch size were varied, the most important settings were observed to be learning rate and epsilon. Setting epsilon to a low value limits the agent's exploration (whether it wants to or not). Many difficult situations were encountered wherein the arm would become stuck in a bad position and vibrate. Increasing the minimum epsilon substantially reduced this tendency. Learning rate was set to .01 for most of the testing. Increasing learning rate sometimes appeared to help but the runs never ended well. Earlier testing had suggested that lower learning rate was ineffective, but after epsilon was adjusted, performance was clearly improved with learning rate reduced to .001.

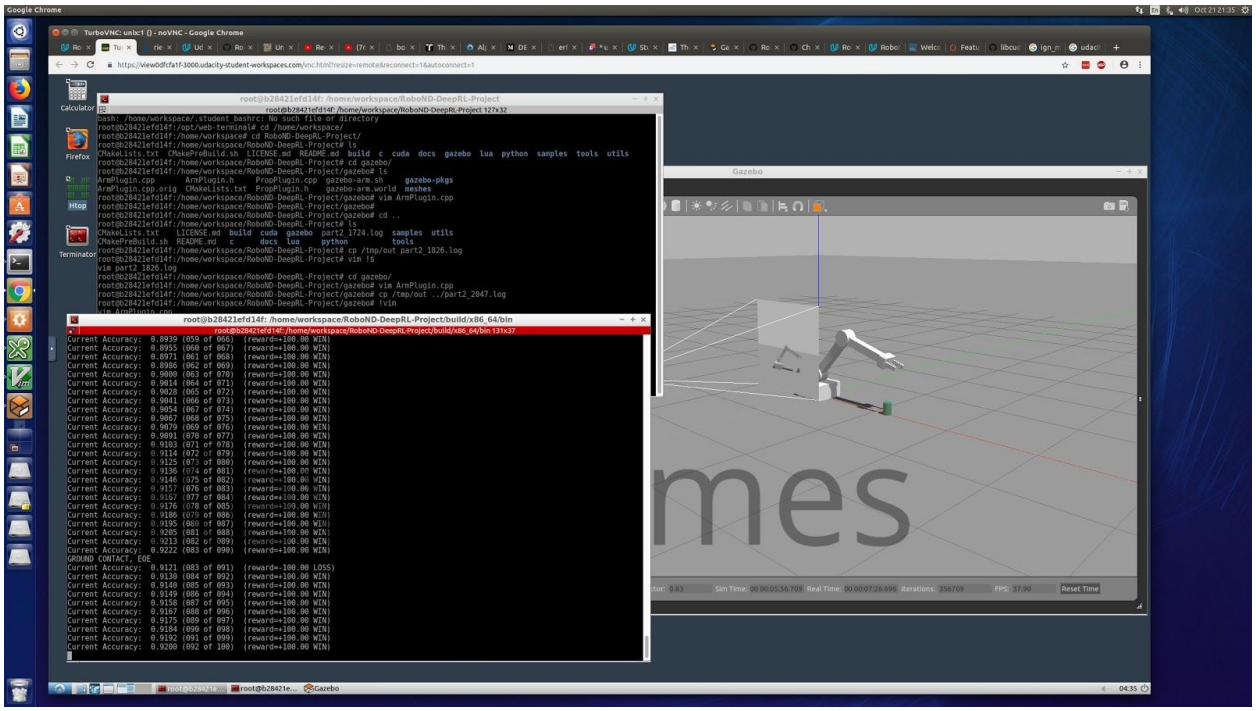
Parameter	Description	Value
Optimizer	Gradient descent optimizer	'Adam'
Replay Memory	Size of past state/action memory	10000

Batch size	Size of batches sampled from memory and used to train network	32
Learning Rate		.001
Gamma	Decay rate in reward with time	.999
EPS start	Initial epsilon; tunes random action probability	.9
EPS end	Epsilon min	.05
EPS decay	Rate of decay in epsilon	200
Use LSTM		false

Results

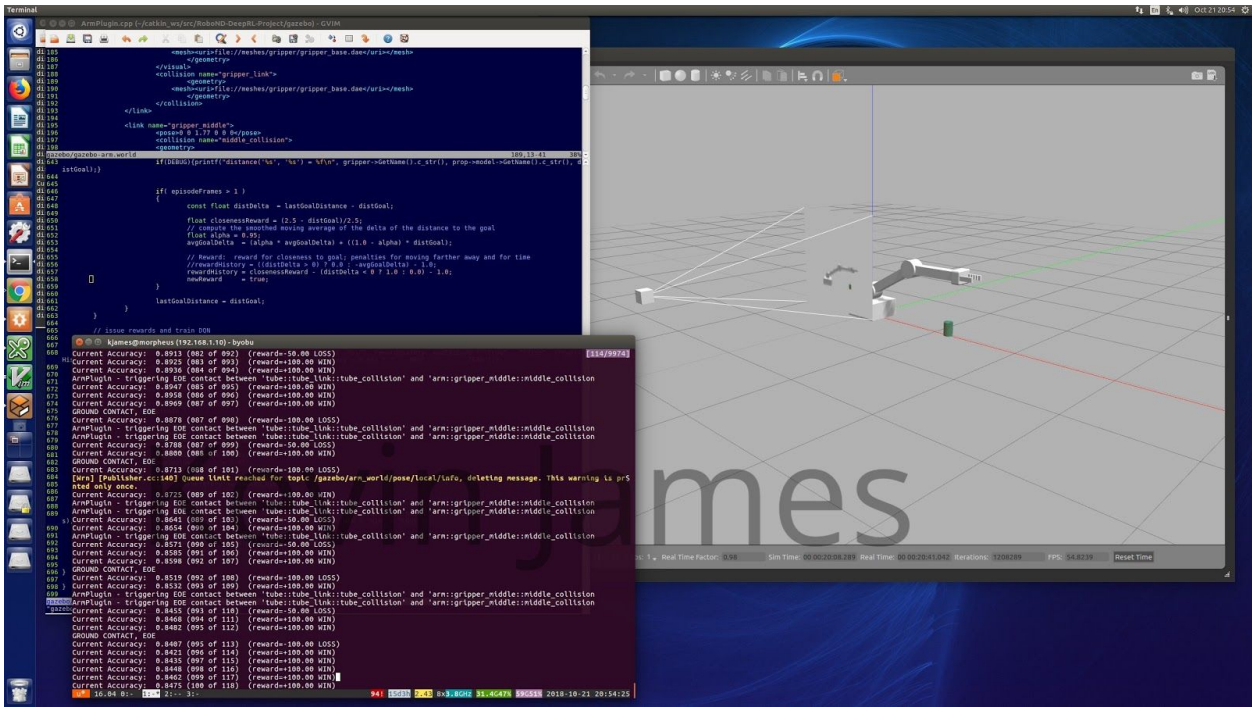
Task 1: First Contact

As shown in the image, the agent achieved an accuracy of 0.92 over 100 episodes of training.

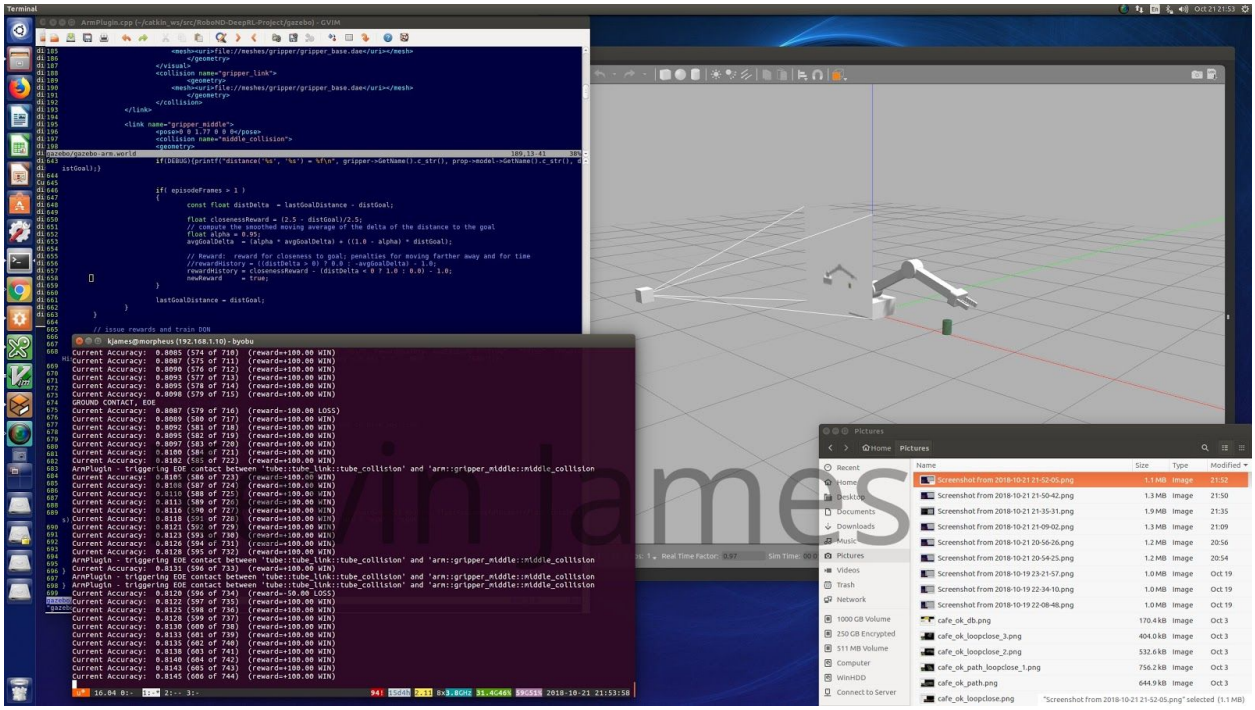


Task 2: Specific Contact

Training for contact with only a specific link was much more involved and had a tendency to require large numbers of episodes to hit the required accuracy. If the agent gets off to a slow start, its initial errors require many successful episodes to be compensated in the average. The screenshot below shows a good run that achieved an accuracy of 0.88 after 100 episodes.



This same training session experienced declining average accuracy over the next 100 episodes. Shown below is a much later point in the session that again exceeded 80% accuracy.



Conclusion