Deep Reinforcement Learning Project Writeup

Gil Meiri

1 Introduction

The goal of the project was to learn about Deep Reinforcement Learning. A robot arm was tasked touching a can that was placed a certain distance away, with its gripper. The robot arm consisted of three actuators, one at the base and two joints in the arm for a total of three degrees of freedom.

Our agent was given as an input a video feed of a camera that was positioned offset. In turn our robot had to use reinforcement learning to learn which action to take at any given state. The actions consisted of movement of an actuator right or movement left for each of the actuators for a total of six possible actions. To make the task easier the base was deactivated so in essence there where really four possible actions.

In this project we had three tasks.

1) Turn the action commands issued by the agent into actual robot movement. For this we had choose one of two choices. The commands could either increase or decrease the actuator velocity or it could change its absolute position.
2) Create a reward function every given state. This functine is used to give a reward back to the agent after every action it takes for the purpose of learning.
3) Tune the agents learning hyper-parameters.

The project was broken down into two tasks. The first was to have the robot arm touch the object with at least a 90% accuracy for a minimum of 100 runs. The second task was to have arm's gripper base touch the object with at least a 80% accuracy for a minimum of 100 runs.

2 Reward Functions

The joint control that was implemented in the robot was position control. This meant that for each action the appropriate actuator would either increase its angle or decrease it. We did this because we wanted the state space to be smaller and only influenced by the robot arm previous configuration and not by the set of actions that got it to that state. We thought that the reduced state space will make it easier for the agent to find an good solution faster.

Choosing a good reward function was crucial to getting the robot to learn. In this project we used three types of rewards.

1) Win Reward - This reward is issued when the robot completes its given task. After this reward is issued the season is terminated
2) Loss Reward - This is a penalty given to the robot if it reaches a state will cause the season to terminate, but the robot has not yet achieved its goal.
3) Interim Reward - This is a reward given in every other state to help guide the robot in the correct direction.

We found the proportionate size between these different types of rewards was important to performance. We also found that the absolute values where also very important. We will elaborate more in section 2.4.

2.1 Win Reward

For this project the win reward was given when the robot achieved its goal of touching the can. For the first task this was with any part of the arm. For the second task this was only of the gripper base. The value we choose for success was very high compared to other rewards. By doing this we made the model gravitate towards choosing actions that will move it towards this winning state. By choosing such a high value as opposed to trying not to end the episode and collect more interim rewards.

We also chose that the absolute value of the win reward to be very high. The reason for this is that we wanted the agent to train quickly even if it only saw a win a few times.

2.2 Loss Reward

The loss was given when the episode ended and the robot didn't reach its goal. We chose that the episode should end when the robot hit the ground, touched the can or ran more than 100 steps without reaching the goal. The reason for ending the episode ending when hitting the ground was that the robot would break. The reason for ending the episode when the robot touched the can with part of the arm that wasn't the gripper it would move the can from its position. The reason for ending the episode after 100 steps was to discourage loops that would cause the robot to go on forever.

For the first task the loss reward was given only when the robot arm hit the ground or the training ran over 100 steps without reaching the goal. For the second task we also gave a loss reward for the robot arm hitting the can with a part of the arm that isn't the gripper. The penalty we gave for hitting the ground was proportionate to the distance the

gripper was from the can at the time the arm hit the ground. The closer the arm was to the can the lower the penalty. The penalty we gave for the robot arm hitting the can was just a fixed number. The same number was giver if the episode ended after 100 steps.

We tried to keep the value of the loss penalty around one order of magnitude larger than the interim rewards. The reason is we wanted the model to gravitate away from these states but didn't want their effect to propagate too much since in some cases when the robot robot arm is close to the can one type of action would cause the arm to lose (i.e. hit the ground or hit with can with the arm) but another action will cause the robot to win (hit the can with the gripper). Since these states can still cause a win we still want the robots to reach these states and not gravitate away from them. On the other hand we do want these values to be larger than the interim rewards so the robot will prefer to move away from a risky situation even if it imposes a penalty as apposed to ending the episode.

2.3 Interim Reward

For the interim reward we checked to see difference between the distance between the gripper and the can in the last state to that of the current state. We used this measurement to give one of types of rewards. If the arm didn't move we gave a fixed penalty. If the robot arm moved we gave it a reward or penalty that was proportionate to this difference in the distance.

2.4 Reward Value sizes

As stated before the relative size of the reward values between the three types of rewards is important, but also their absolute values. When an agent is initiated it is given random weights which gives it random behavior. When given and action a predicted reward is returned which not exact but still have a value. This encourages the agent to explore the action space. If the values of the real rewards are much larger in magnitude of this random prediction the new samples will quickly "overpower" the random predictions and change the model. This discourages new exploration. Even though the agents has an EPS parameter tells the agent to choose random actions from time to time, this value diminishes over time. We found that keeping the rewards relatively small (apart from when giving a win reward) caused the agent to take more random actions, especially in the beginning when it is starting to learn. From our experience this was more effective than the agents EPS.

3 Hyperparameters

Tuning the hyperparameters was challenging. Changing some of the hyperparameters had little to no effect. Others where essential to getting the agent to learn properly. The same hyperparameters were used for both tasks. The list of hyperparameters and their values is given in table 1.

Changing the GAMMA parameter which controls how much weight to give to the current reward as opposed to future expected rewards didn't make much of a difference when it came to performance. Same can be said about the EPS variables. The INPUT_WIDTH and INPUT_HEIGHT were reduced. This made a big difference in performance. This made the input space of the neural network much smaller which allowed it to train easier. The NUM_ACTIONS was chosen to be 4 which is not the 6 actions the robot is capable of since in our exercise the robot is unable to move the base actuator so there is no point in the robot learning that these actions are useless. This made the agent easier to train. The optimizer we chose was RMSprop. We also tried Adam but saw no major difference in performance. The learning rate was unchanged and so was the replay memory. BATCH_SIZE was increased. This made a huge difference. The LSTM didn't make much difference if it was on or off.

TABLE 1
Agent's Hyperparameters

Hyperparameter	Value
INPUT_CHANNELS	3
ALLOW_RANDOM	true
DEBUG_DQN	false
GAMMA	0.9f
EPS_START	0.9f
EPS_END	0.05f
EPS_DECAY	200
INPUT_WIDTH	64
INPUT_HEIGHT	64
NUM_ACTIONS	4
OPTIMIZER	"RMSprop"
LEARNING_RATE	0.01f
REPLAY_MEMORY	10000
BATCH_SIZE	256
USE_LSTM	true
LSTM_SIZE	256

4 Results

With the correctly tuned hyperparameters and reward functions the agent was able to learn for both tasks. The results for the first task can be seen in Figure 1. The results for the second task can be seen in Figure 2.

In both cases the robot seemed more explorative in the beginning and had allot of loses in the beginning, but started to get consistent wins the more it ran. Sometimes it seemed that the robot was bit too explorative and it took along time for the robot to get consistent wins even though it had allot of sample of routes that concluded in wins. Tuning the reward function absolute values can probably remedy this.

An interesting case that came up allot when the absolute values of the reward function were too high was that the agent would learn to lose quickly. The agent would find a path that would maximize the interim rewards, meaning that the arm would go towards the can, but then terminate the episode by either hitting the ground or the can with a part of the arm. The agent would run many times but wouldn't try to hit the can with the gripper a single time. This meant that the the agent had no knowledge of that there was a really big positive reward in the action space. So since the agent was trying to maximize its reward, it turned out that it was maximizing its interim reward. It did not matter how negative the loss reward was since the agent had no idea that terminating with a negative

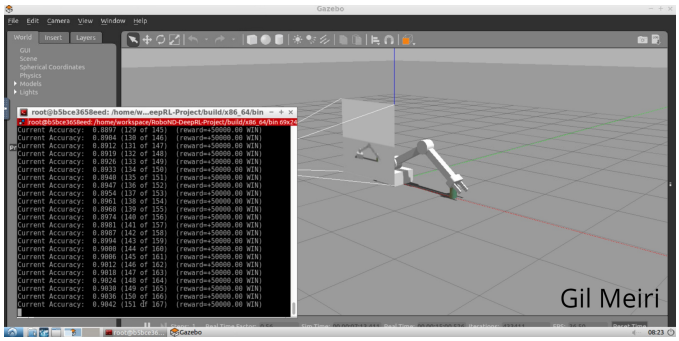

Fig. 1. Screen capture of the robot completing task 1

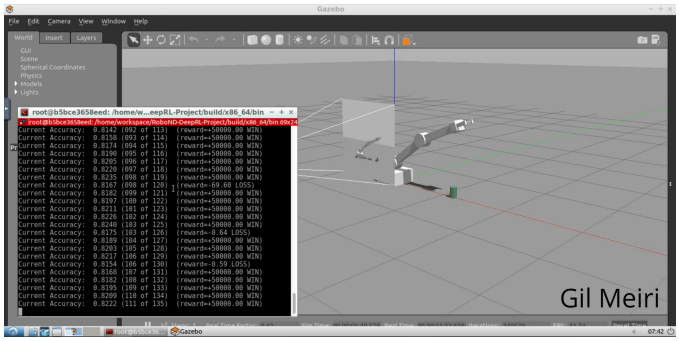

Fig. 2. Screen capture of the robot completing task 2

cumulative reward was a bad thing that was to be avoided. Even though EPS was supposed to remedy this situation, but EPS works randomly and there is a much greater chance that the random action would happen somewhere in the beginning of the path where the gripper is far from the can. This wouldn't make much difference because the arm would move in a slightly different path that quickly converged with the original path. As mentioned above, changing the absolute values of the rewards changed that and made the changes in the agents neural network slower made the agent more explorative. The only exception we made was we wanted the win reward to be very large so that if the agent finds a path that caused it to win it would try to replicate that path.

5 Future work

The subject of deep reinforcement learning is a very challenging and interesting one. One interesting thing to research would be how the size of the absolute values of the reward effects the exploration vs using the optimal path that has been found.

One area that might be interesting to explore and might yield better results is to give a lower reward when the episode ends as a result of running more than 100 steps as this would avoid the situation we discussed where the try to grab as much interim rewards as possible and end the episode as quickly as possible. This might encourage the agent not to terminate so quickly and allow us to increase the absolute values of the other rewards so we are less reliant on the random actions of the untrained neural network. This would probably cause the agent to convert to optimal policies faster without the need to explore more of the action space.