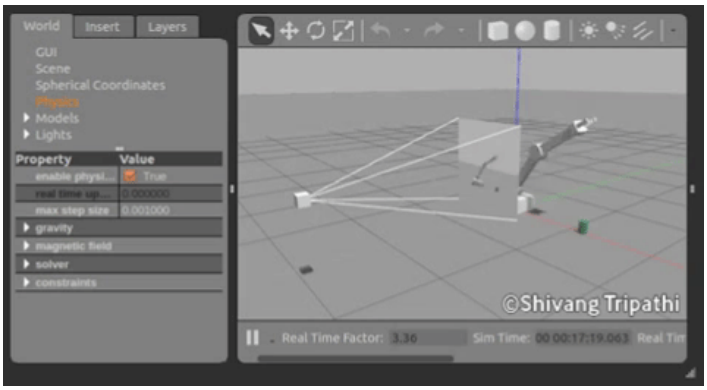
Report WriteUp

This is the writeup for the Deep RL project based on the Nvidia open source project "jetson-reinforcement" developed by Dustin Franklin. **The goal of the project is to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:**

> 1. Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.
> 2. Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy.

Hyperparameters tuning has the most challenging and time taking part of the project.



Reward Functions

The design of the reward function is the single most important task to accomplish any objective using reinforcement learning. Spending time on designing a good reward function saves a lot of time in later training and tuning the hyperparameters. The agent always tries to optimize the cumulative reward which can sometimes lead to undesired unplanned behavior. This is also called the Cobra effect.

The distance from the gripper to the goal prop was measured using the `distGoal(gripper, prop)` method. Change of distance to goal was called `delta = distGoal - lastGoal` . As evident, negative `delta` implied the gripper was moving towards the goal pose.

Exponential moving average of the delta `avgGoalDelta` was used in the reward function.

```
alpha = 0.7;
avgGoalDelta = avgGoalDelta * (1 - alpha) + delta * alpha;
```

Higher `alpha` value meant lesser weightage to the previous values of distribution compared to the later values.

```
reward = -C1 * avgGoalDelta
// -C1 is a negative constant
```

Its important to note that this is a positive reward function, i.e., as the gripper moves towards the goal prop, the rewards will be positive.

The important point to note in the case of positive rewards is if the agent can accumulate excessive positive reward without termination *by simply staying in its place*. This does not happen because if the delta stays **0** (position not changing), the rewards will (*smoothly*) become **0**. Thus, training with this reward function will not lead to the robot getting stuck to a position and still accumulate large rewards.

The faster the robotic gripper moves towards the prop, the more the rewards will be. This is because the value of the `delta` will be higher. Thus, this reward function incentivizes the fast robot gripper and prop collision.

Hyperparameters

The `INPUT_WIDTH` and `INPUT_HEIGHT` was kept equal to the size of the input image, i.e., `64 x 64` .

`Adam` was tried with various parameters but `RMSprop` turned out to be better and was used to achieve the required accuracy for both the tasks.

Learning rate was set high enough to learn fast enough to reduce the number of runs required to achieve the required accuracy. With higher rates, the accuracy was not reached and with lower rates, the number of runs needed increased by order of hundreds.

The degree of randomness though was set to `true`, was done with a low `EPS_END` of `0.005` to prevent the agent the agent from trying random trajectories every so often, taking more time and runs to reach required accuracy.

Larger batch size leads to convergence to narrow goal cases. This means that the agent will work only for cases in which the prop is in the narrow neighborhood of the prop training region. The agent will fail to generalize for the wide range of goal positions that it may encounter. Thus, a smaller batch size was used to generalize for various distance goal instances.

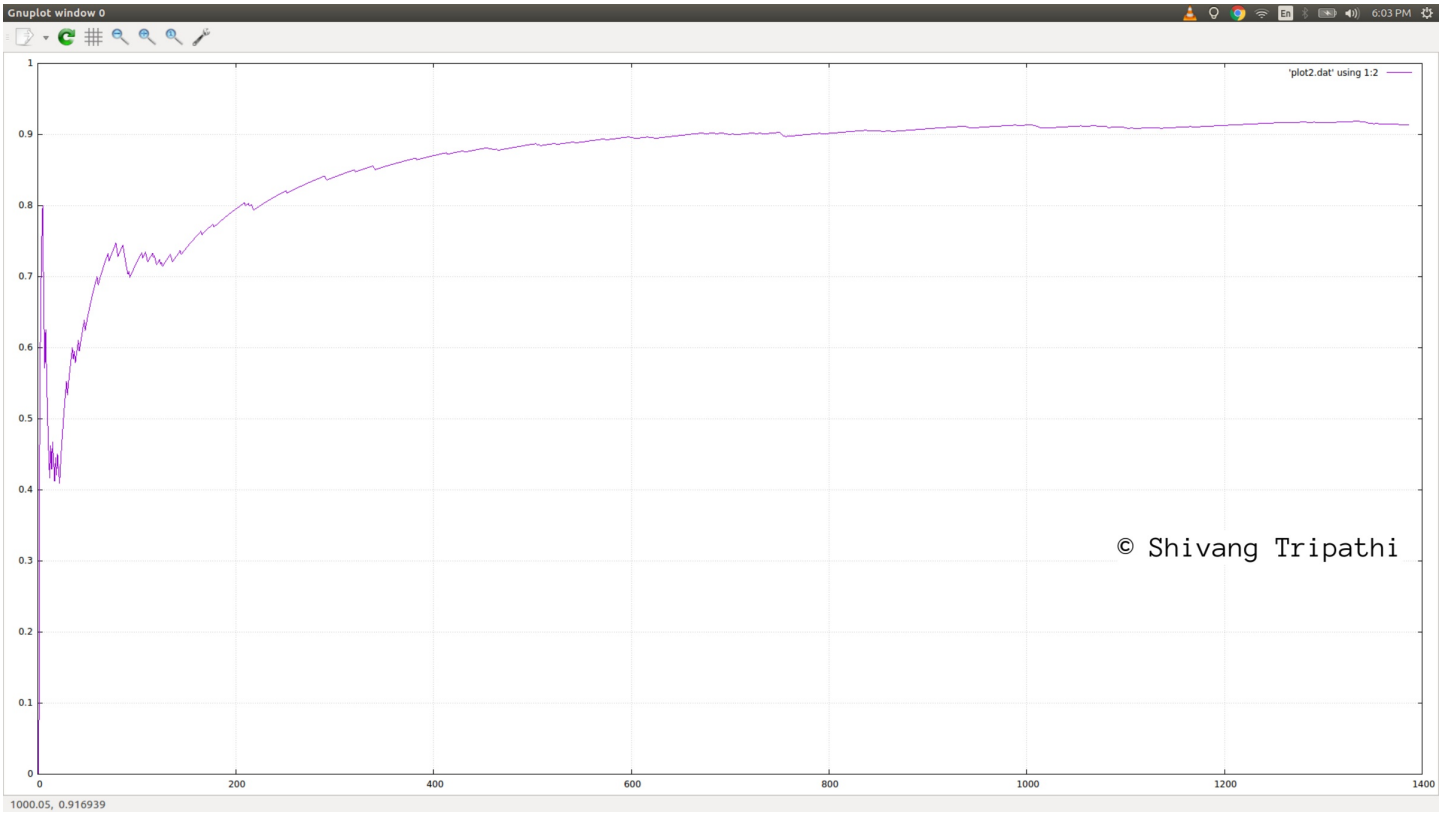
The LSTM size was chosen depending upon the length of sequence that will need to be learnt. Episodes terminated at 100 runs but the agent was able to come to a conclusive result(ground or prop collision) by the 30 - 40 episode. Also, making size a multiple of 32 may actually speed up the training by a small amount. Thus, a value of 64 was used for this task.

Results

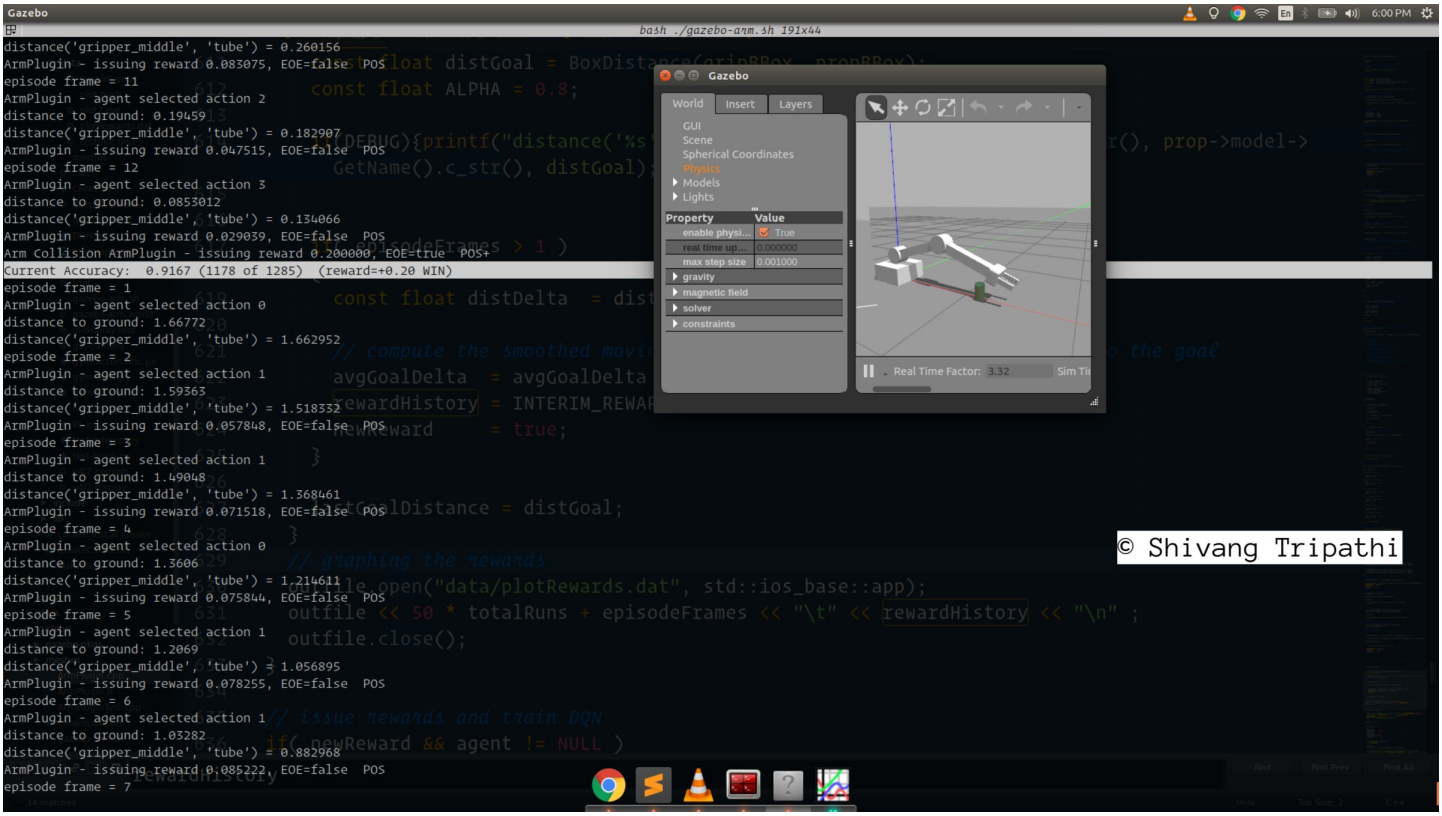
The first task to achieve was:

> Getting the robotic arm to touch the goal prop with more than 90% accuracy.

The watermarked screenshot of the **task 1 accuracy graph**:



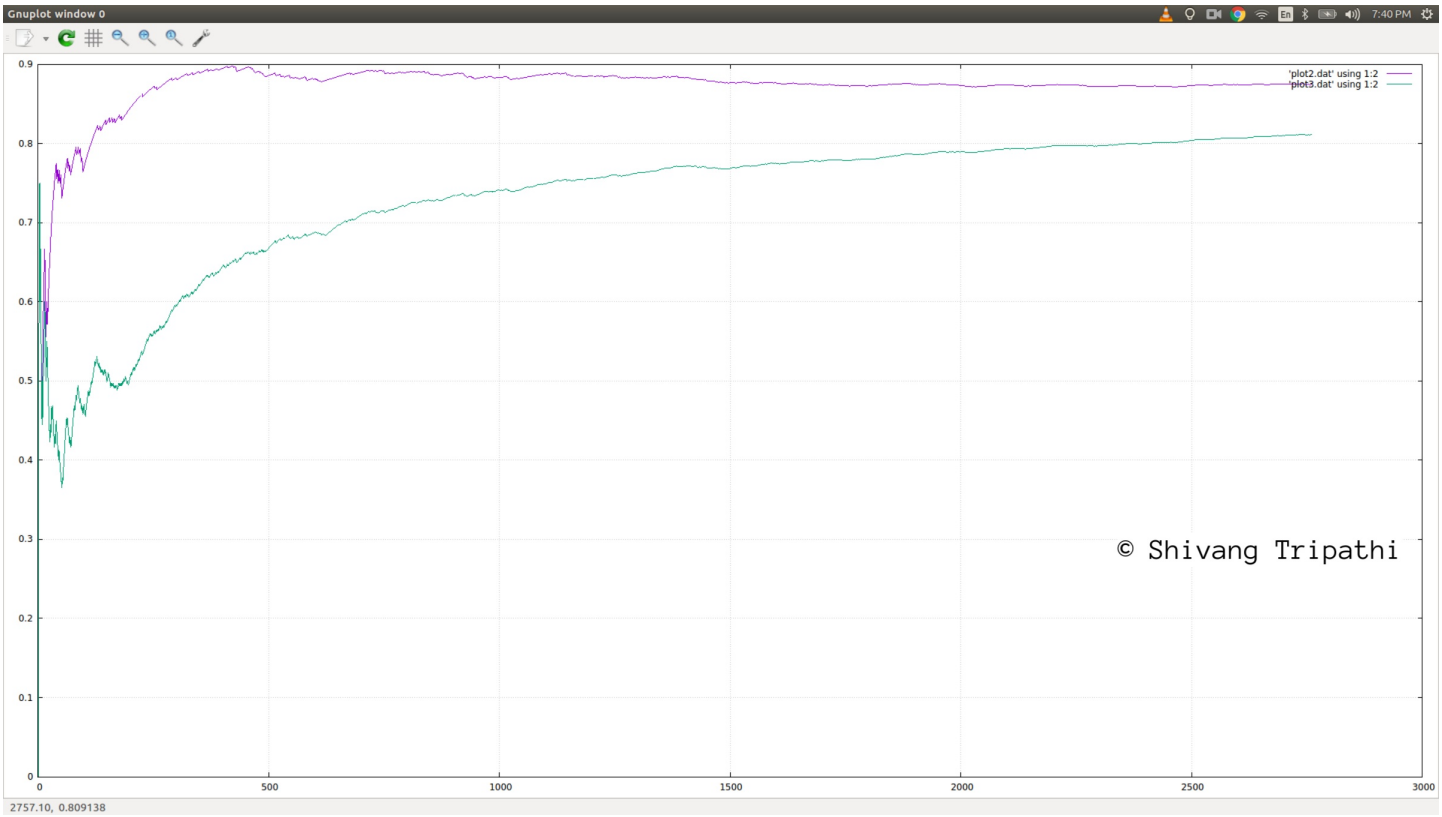
It took close to `700 iterations` to reach the `90% accuracy` mark. It reached to `91.67%` by the `1285th iteration`.



The task 2 was more refined:

> Getting only the robotic gripper to touch the goal prop with more than 80% accuracy.

This took more time and runs but was a more meaningful task to accomplish for the agent. It touched the `80% accuracy` mark after around `2400 iterations`. The watermarked screenshot of the **task 2 accuracy graph**:



By the `2808th iteration`, the accuracy was at `81.23%`

Future Work

To improve the current results,

- Find a better reward function.

In the current reward function, until the very end of episode, the agent is blind as to where it is heading. At the very end, it may encounter a collision with a gripper(high reward) or a ground(high penalty). The reward function should also consider the current distance between the gripper and the ground.

- Bright coloring the gripper and the prop.

This will make it easier for the CNN in DQN algorithm to find the relation of gripper and prop position with the reward received.

DQN is a novel end-to-end reinforcement learning agent that can perform diverse range of tasks with superhuman level of mastery without human intervention. This is a major technical step in the quest of general AI. Paradoxically, the algorithms that power this machine learning are manually designed. Instead, if this learning could to automated, this could open up new opportunities. Learning to optimize and Learning to Learn by Thrun & Pratt, 2012) was a step in this direction which I would like to implement as they used RL to learn the optimizer.