Deep Reinforcement Learning Dodgeball

Kevin Tan and Andy L. Khuu

Introduction

An exciting area of research for DRL (Deep Reinforcement Learning) is to develop digital agents that eventually get deployed into physical robots, a task which OpenAI's Dactyl project [6] demonstrates requires high-fidelity training environments. In this project, we designed a physically-realistic simulation with high-dimensional sensory data sources, and trained an agent using a refined deep Q-network in it.

Environment

Our environment consists of a room with agent and opponent cubes and balls. At any given state S_t, the agent can take one of seven actions A_t: stay still, rotate clockwise/counterclockwise, and move forward/backward/left/right. The environment advances by converting agent actions into forces and torques applied to the agent's physics engine representation, then stepping through the simulation by one frame.

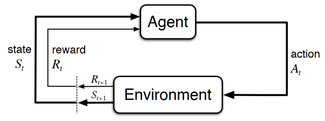


Fig. 1: Agent-Environment Interaction Loop

After our agent chooses an action A_t, our environment returns the next state S_{t+1} and reward R_{t+1}. We experimented with having our S_{t+1}'s be either a vector of positions/orientations/velocities of the objects in the environment or be simulated LiDaR rangefinder measurements. Our R_{t+1}'s are summarized in the following table, omitting specific values.

Event	Reward
Agent gets hit by opponent ball	Large negative
Opponent gets hit by agent ball	Large positive
Episode length increases by one	Small negative

Agent

Our agent was powered by a refined deep Q-network (DQN) [1] that took as input the state representation given by the environment S_t and computed seven different Q-value predictions corresponding to the different actions available to the agent.

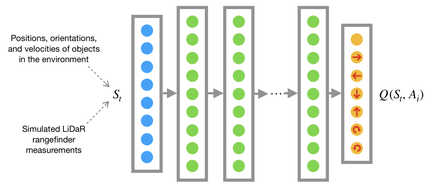


Fig. 2: Our Refined DQN Architecture

Implementation

We used the Unity game engine with the ML-Agents toolkit to build our environment and PyTorch to both define our policy network and update its network parameters.

Experiments

In each of our experiments, we trained our agents for a total of 1000 episodes, set the exploration rate to be 0.2, discount rate to be 0.9, learning rate to be 0.05, and used Stochastic Gradient Descent with momentum as our learning algorithm. During the training process, we used Tensorboard to track the following three training metrics; the first two tended to be extremely noisy while the last was more stable.

1. **Average Total Reward**: the average total reward for k training episodes.
2. **Average Episode Length**: the average episode length for k training episodes.
3. **Average Predicted Q**: sample a random set of states before training and periodically evaluate the average value of those states—according to the DQN—during training.

During the training process, we found that it took an extremely long time for our agent to converge to an optimal policy, and so we devised many optimizations to our basic refined DQN in order to achieve better sample complexity and faster convergence times.

Optimizations

We pursued a number of different strategies in an attempt to help our agent converge to an optimal strategy faster, from algorithmic additions to environment modifications.

1. **Curriculum Learning**: We broke down the ultimate task into smaller sub-tasks which were progressively more difficult. For instance, an earlier task featured a static opponent while a later task featured a predictably moving, hard-coded opponent.
2. **Prioritized Experience Replay**: Starting with a vanilla experience replay buffer [5] in order to both reuse experiences more than once and break the highly correlated nature of successive experiences, we further implemented a prioritization strategy [3] that sampled experiences according to their temporal-difference error.
3. **Environment Complexity Reduction**: To decrease the difficulty of the task required of our agent, we made a variety of environmental changes including reductions of the action and state space as well as modifications to the observational input.
4. **Target Network**: In order to improve the stability of our learning algorithm, we included a target network [4] whose parameters would be periodically frozen versions of the policy network. We found that this partially solved the problem of prolific negative rewards.

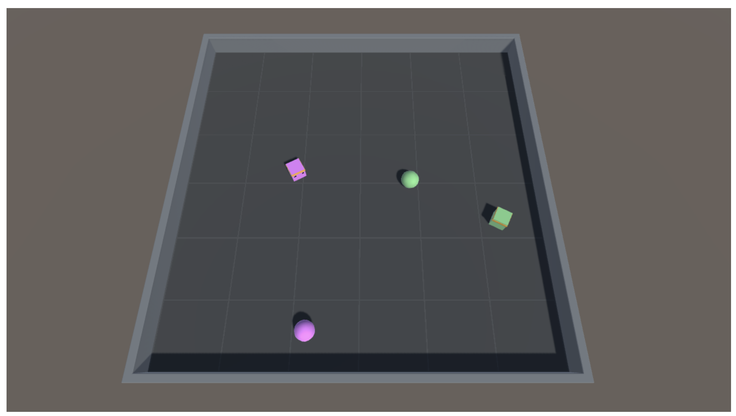
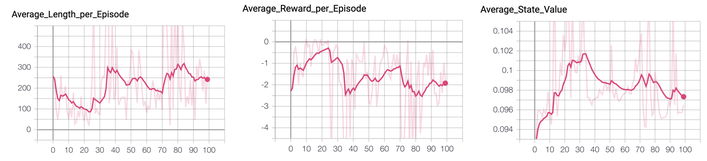


Fig. 3: Our Environment Simulation

Results

The data below comes from training our agent in the ML-Agents Basic environment. The average episode length and reward were not great indications of our agent's performance due to the stochasticity of our exploration strategy. The average state value of a randomly sampled set of states increased more stably, which is similar to what happened in [1]. The drop in average state value at around epoch 30 can be explained by the tendency for Q-learning to overestimate Q-values.



When testing on our built dodgeball environment however, we were unable to achieve similar successes and faced stagnation in the training progress. Due to the lack of interesting results and marked improvement in any of the training metrics, we have omitted these results.

Conclusion

Our method demonstrated strong results in an environment with simple state and action spaces, but was unable to replicate this success in our dodgeball environment—whose state and actions spaces were significantly more complex—despite many optimizations to improve sample complexity. Noting that environments similar to ours, in particular, OpenAI's Hide and Seek environment [2], required 2.69 million episodes before any intelligent behavior emerged, we partially attribute this result to a lack of computation power and training resources; another possible reason is the simplicity of our policy network architecture, which may not be expressive/deep enough to simultaneously interpret raw sensory information and suggest optimal actions.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. ArXiv e-prints, December 2013.

[2] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019.

[3] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. International Conference on Learning Representations, 2016.

[4] van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094–2100. AAAI Press, 2016a

[5] Lin, Long-Ji. 1992. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching." Machine Learning 8 (3): 293–321. https://doi.org/10.1007/BF00992699.

[6] Andrychowicz,Baker,Chociej, et al Learning Dexterous In-Hand Manipulation." n.d. Accessed December 2, 2019. https://arxiv.org/abs/1808.00177.