Neural.Swarms and Neural.Orb

Tabor C. Henderson, Galvanize Denver Data Science Fellow
thetabor (at) gmail (dot) com, github (dot) com (slash) thetabor

Deep Quality Networks

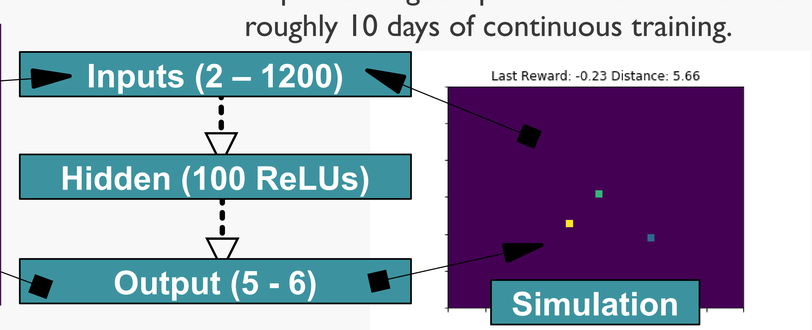
- Deep-Q Networks apply reinforcement learning to neural networks.
- The network predicts Q-values for each action the network is allowed.
- A Q-value represents quality of a state, or the expected sum of rewards as we play the game from that state.
- Our agents (almost) always select the max Q-value we predict.
- DQN's were deployed to great effect by DeepMind, most recently in their Go-playing AI, AlphaGo.

Objectives

- Simulation: demonstrate DQNs
- Navigation: utilize DQNs in a physical space
- Coordination: train multiple DQN-based agents to cooperate

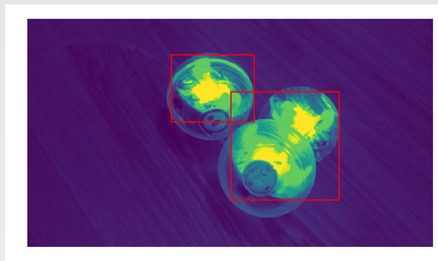
Simulation

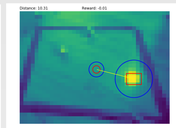
I developed the Neural.Swarms simulation engine in which I can deploy either supervised or reinforcement learning. The game is a simple grid, with an agent moving one square at a time to a goal. Below is an agent (yellow) in an environment with a goal (teal) and an obstacle (blue). My best agent had a simple MLP architecture with one hidden layer containing 100 ReLUs and hyperbolic tangent output units.

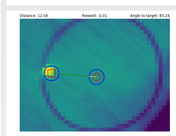


Reinforcement Learning

Reinforcement learning algorithms learn from feedback in their environment. This requires a system of providing that feedback, which was challenging in both simulation and physical spaces. Once a good system of rewards was in place, I could document learning. The simulated agents performed well given sufficient training episodes (about 160,000).



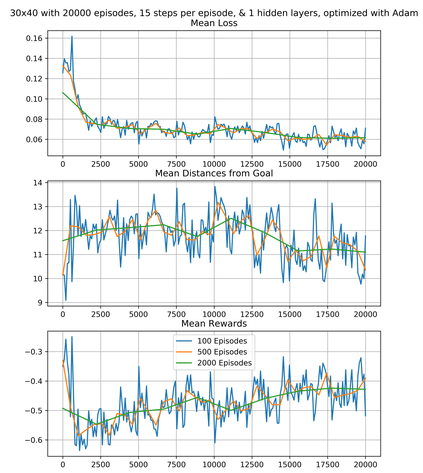




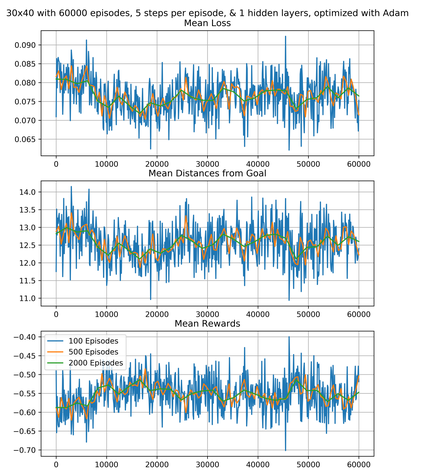
Deploying to Sphero

- Deploying a DQN to Sphero required a computer vision interface, and controlled training environment.
- SciKit-Image provides excellent filters and segmentation modules, and pygame.camera allows access to webcams.
- Bluetooth instability frequently interrupted training runs, and at one point the host machine overheated and crashed.
- The Sphero DQN couldn't utilize the experience replay technique, which has been shown to accelerate learning.
- Each game step took 4-5 seconds, meaning performing the million steps required for good performance would take roughly 10 days of continuous training.

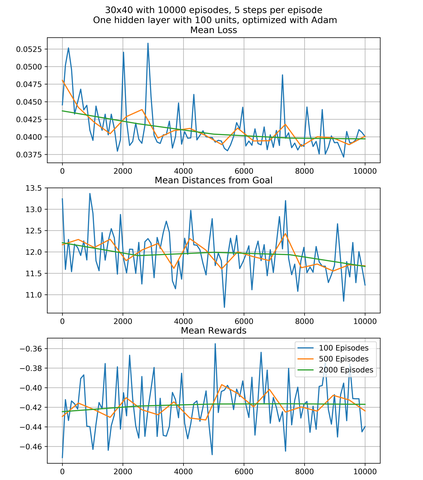
Break In



Full training



Harder Task



Measures

Measuring the DQN's learning was difficult, as they require a massive amount of training episodes (similar to epochs in supervised learning). I used three different measures of network learning

- Loss: mean squared error, commonly used for many machine learning algorithms
- Mean distance to target: measured over one or more game episodes, this provide feedback on actual performance in game
- Mean reward: measured in the same intervals as mean distance, mean reward tells us the same basic information as mean distance, and their inverse relationships confirm that the scoring system works.

Accelerating Training

I accelerated the model training with several techniques:

- Explore/exploit function
- Guided training
- Deterministic strategy integration

Conclusion and Next Steps

The training program works, but simply doesn't run fast enough for me to have produced a viable model in the allotted time frame. However, a viable model is possible given a few improvements:

- Implementation of experience replay for the physical model
- Simulation engine similar enough to the physical problem that we could pre-train the model virtually, then deploy
- Utilize all data from the Sphero's on board sensors

References

1. *Andrej Karpathy*
2. *Eder Santana*
3. *Charles Isbell, Michael Littman, and Chris Pryboy via Udacity*
4. *Tambet Matiisen*