ROB 537 Homework 4: Learning-Based Control

Nathan Butler | butlnath@oregonstate.edu

Academic Statement

Unless otherwise stated, all work in this assignment was my own. I worked with Noah Boehme on identifying refences for the DQN in Part 1 and for mutation types in Part 2.

Q-Learning Agent

A Deep Q Network was used to address the issue of continuous state space for the CartPole problem. In this DQN solution, a state representation is fed into the network, which then outputs Q-values for all possible actions that could be made from that state. References [1] and [2] provided helpful examples of DQN architectures for function approximation, and the training architecture used in [3] was effective for my implementation.

The DQN implementation also features a replay buffer to address catastrophic forgetting and an identical Q-network used in training to address instability per the recommendations in [1]. The replay buffer stores a history of "experiences," which include a past state, the action taken from the state, and reward gained, the next observed state, and whether the simulation failed. In training, batches of experiences are pulled randomly from the hundreds of states stored in the replay buffer so that the network trains on a wide variety of experiences. The second "target" network introduces greater stability by acting as the ground truth during training. In implementation, the value of $\max(Q(s_{t+1}, a))$ is received from the target network. Periodically, the target network's weights are synchronized with the Q-network's weights (which have theoretically improved since the last synchronization) and training resumes.

The network architecture was revised over much testing and experimentation, ultimately resulting in a network featuring two hidden layers with 64 neurons each. This network was trained over 10000 episodes, using an epsilon-decay strategy to balance exploration with exploitation during the first half of training. The target and Q-networks were synchronized every 1000 episodes, and the network was given a large experience replay buffer to draw from. The results of training are displayed in Figure 1, which shows that the DQN achieved consistent performance of approximately 300 time steps by the end of training.

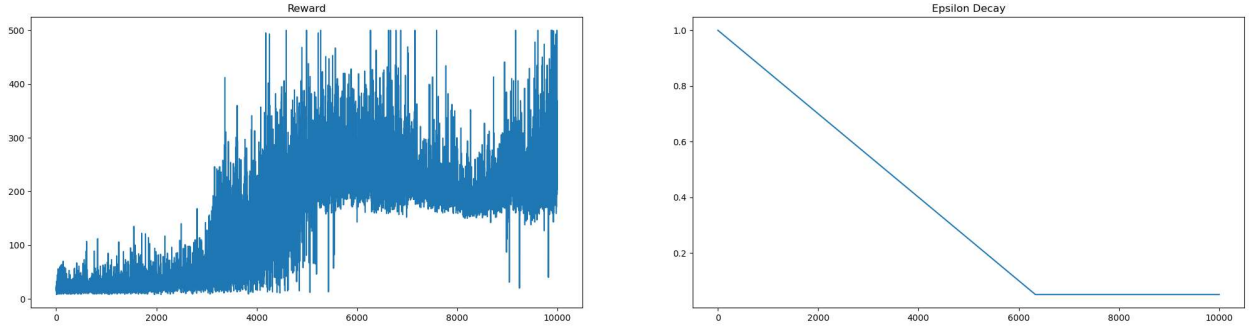

Figure 1. DQN performance and epsilon decay during training

Evolved Neural Network

The solve the CartPole problem using a neural network, a different architecture was implemented. In this approach, a population of "agents" is generated, each with a Q-network for producing Q-values from state representations as described in part 1. While the hyperparameters are held constant across all agents, the weights within the networks are randomized initially and mutated during the evolutionary process. The algorithm works as follows:

1. Generate a population of k agents and evaluate their fitness by running each in several simulations and logging the average reward gained by each agent over their simulations.
2. Now for each training cycle, first increase the size of the population to 2*k by either mutating existing members of the population or, on rare occasions, performing crossover between two parents in the population to generate two children. Mutation and crossover operators selected from the following list [4]:
    a. Mutation operators (performed on up to 50% of the parameters in a layer):
        i. Replace a random weight with a random value (-1 to 1)
        ii. Scale a random weight by a random value (0.5 to 1.5)
        iii. Subtract a random number between 0 and 1 from the weight
        iv. Flip a sign of a random weight
    b. Crossover operators:
        i. Swap the weights of two random neurons in a layer
        ii. Rarely, swap two layers
3. Next, for each member of the population not previously evaluated, run that member in several training cycles and record the average reward gained (large bonus awarded if the training is truncated).
4. Use roulette wheel selection based on the average reward gained by each agent in the population to select down to k. Repeat to 2.

Figure 2 shows the average training performance of the population and the fitness of the best-found solution over training.

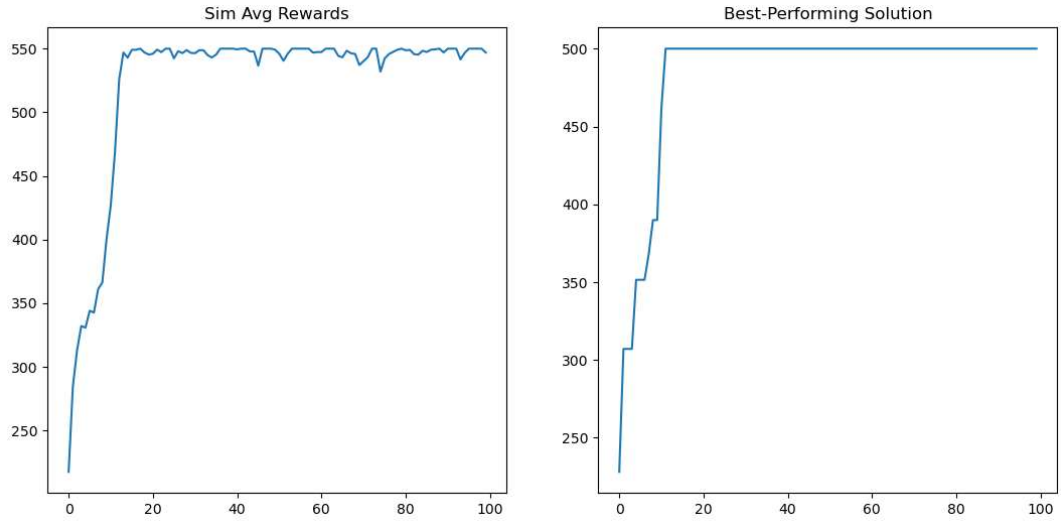

Figure 2. Average neural network performance and best overall performance during training

Discussion

Both algorithms have advantages and disadvantages. Beginning with DQN, this algorithm is beneficial in that is can actively adapt and learn during runtime using both current and past experiences. However, training with somewhat sparse rewards can be challenging, and issues such as catastrophic forgetting and training instability are additional hitches that need to be addressed. In attempting to train this algorithm, I found it challenging to tune the replay buffer size and synchronization frequency parameters in a way that yielded improvements over training. Ultimately, storing larger batches of experiences and synchronizing less frequently seemed to have better results, but these changes required greater training time.

On the other hand, the evolutionary algorithm has a more straightforward implementation and evaluation, but cannot learn online and is computationally expensive to train. Testing and evaluating populations of randomly-tweaked networks is also a much less direct approach than the backpropagation used in Q-learning. I found for this strategy that a simplified network could be improved in more meaningful ways than the larger two-hidden layer network used for the DQN approach.

Overall, both algorithms produced networks that performed well in the CartPole environment, with the evolutionary algorithm finding high-performing solutions more rapidly than the DQN.

References

[1] M. Polkhan, “Building a DQN in PyTorch: Balancing Cart Pole with Deep RL,” Medium, 2020 [Online]. Available: https://blog.gofynd.com/building-a-deep-q-network-in-pytorch-fa1086aa5435

[2] T. Miller, “Introduction To Reinforcement Learning,” github.io, 2022 [Online]. Available: https://gibberblot.github.io/rl-notes/single-agent/function-approximation.html

[3] Alex, “Cartpole-v0 loss increasing using DQN answer,” stackoverflow.com, 2019 [Online]. Available: https://stackoverflow.com/questions/56964657/cartpole-v0-loss-increasing-using-dqn

[4] remode32, “How to evolve weights of a neural network in Neuroevolution?” stackoverflow.com, 2015 [Online]. Available: https://stackoverflow.com/questions/31708478/how-to-evolve-weights-of-a-neural-network-in-neuroevolution

Appendix

Link to github code repository: https://github.com/natbut/rob537_hw