Brandon Rose

CS 360

12/4/2024

Reinforcement Learning Writeup

For this assignment I created a DQN to play and optimize solutions to a GateGame. The game itself has a agent choosing between gates, taking steps up, down or remaining in the same spot in an attempt to maximize their score which is called reward. My program begins by initializing the hyperparameter swhich are used to train the model. Then it defines the three DQN networks that will be trained, in my case I did a small one which is 64x64, a medium at 128x128 then a large at 256x256x128. The DQN class takes in layers, the input features of the game, and the number of actions which can be performed. Using the layer sizes, it creates layers separated by a Relu activation function to introduce nonlinearity. The forward function of the class takes a state and returns Q-values for all actions. While the DQN is not initialized in main, it is done in the train DQN function which creates a GateGame object initializes the DQN and then performs training of that DQN. The training is done over a series of episodes, and for each episode it’ll reset the GameGate object to the initial state play the game and store experience in a replay buffer object.

The replay buffer object allows for sampling of batches during training. It stores past experiences and ensures that training data is independent. Moving on in the train function as it continues through the episodes, the learning rate and exploration rate is adjusted over time with training done in the train step function. At the end of all episodes for the network it logs the results for comparison at the end. In the train step function, a single step of training is performed. It takes in DQN, a target net (which is a stable version of the DQN for computing Q-values), the optimizer which is Adam in this case, and a batch of samples from the replay buffer. The function unpacks the experiences, predicts states on the current network and maximum q values from the target network. As all three networks are trained, they are then compared. I found that the large network with a learning rate of 0.001 seemed to work best.

For the rest of my hyperparameters I trained each model on one-thousand episodes in batches of sixty-four experiences in each training step. My discount factor (gamma) was .99 which encouraged future rewards and learning as opposed to immediate ones. My exploration rate started at 1, with a minimum it could go down to being .1 while decaying at a rate of .995. The target network is set to update every ten episodes ensuring that it wasn’t updated too frequently but still making changes in a reasonable amount of time. I experimented with a variety of hyperparameters and found that a high discount factor performed much better than a low one, and likewise a higher learning rate of around .0001 was better than anything lower for this problem.

This assignment was particularly difficult for me while I was able to find information on DQNs for games there was one website that broke down a lot of the information in regard to building one to play a game much like the GateGame which was extremely helpful. Another scholarly article from Lehigh University also helped me shape my own network and broke down a majority of the process for DQNs which I found to be extremely helpful. It was interesting to see how much tweaking a single hyperparameter would alter the rewards by so much, for example a .001 lr performed nearly three times better than a .0001 lr. Additionally, the effectiveness of each network seemed very volatile. On some runs it would remain in a consistent range of rewards for the last hundred episodes, but in other cases it would jump into the tens of thousands seemingly out of nowhere. While I am sure that this is not supposed to happen, I was unable to figure out why it would be so volatile at times and am assuming that it’s due to the randomness in GateGame.

Overall, I feel like the DQNs turned out well, but there is obviously room for improvement in implementation. In the future I could look to plot out the results to see the improvement of each network as it goes through each of the episodes as well as record other metrics such as cumulative rewards. An interesting feature I found which could be added in the future was reward shaping which would seek to reward the agent for immediate improvements to the score and penalize it for unnecessary moves. I could also continue to fine-tune the hyperparameters for a better DQN, I did a lot in terms of getting them where they are now, but it would be good to continue to experiment with them and potentially find a better combination of them.

Mentioned References:

<https://medium.com/@www.seymour/training-an-ai-to-play-a-game-using-deep-reinforcement-learning-b63534cfdecd>

<https://www.researchgate.net/profile/Afshin-Oroojlooy-Jadid/publication/319209704_A_Deep_Q-Network_for_the_Beer_Game_A_Reinforcement_Learning_Algorithm_to_Solve_Inventory_Optimization_Problems/links/5adca362a6fdcc29358aab1e/A-Deep-Q-Network-for-the-Beer-Game-A-Reinforcement-Learning-Algorithm-to-Solve-Inventory-Optimization-Problems.pdf>