Michael Peter C. Dansereau
CS 360: Artificial Intelligence
Fall 2024

Reinforcement Learning Write-up

As has become habit, this paper will consist mostly of apologies for the various ways in which my program failed to approach anything resembling functionality. As you may have already surmised, it never successfully ran, nor did it come within any reasonable margins of success in any of its too few facets, and so I do not have three networks with internal structures to analyze. Perhaps that complete dearth of internal structure is the primary fault in my implementation. But if there had been three networks, some things I may have tweaked to record the differences in efficacy include the number of repeated games on which the networks are trained. In particular, the variable determining the rate at which repeated networks decays would have been subject to great scrutiny. It is my assumption devoid of reality that were the rate of decay too fast, it would dissolve the odds of repeated games and, unsated, sally forth to break down the ability of the network to learn. With too great a variety too early in its life cycle, the network would, as if a dinner guest of Trimalchio, gorge itself on too many scenarios without being able to truly appreciate or learn from any of them. In other words, it would suffer from underfit. On the other hand, were the decay value too slow, the training set would consist of too little variety, thus summoning underfit's nefarious sibling, overfit. It would also be possible to provide some networks with more or less information in hopes of yielding some mysteriously begotten benefits. To ascertain the correct amount of information to give the network, we must call upon Goldilocks, resident expert in trial an error. Papa Bear might give the network access to the entire network, but his network might look too far into the future and confuse itself with far off data, losing efficacy in the present. Mama Bear might give her network access to only the current set of gates, but she would wind up performing no better than a simple greedy approach. But Goldilocks being fictional and my program non-functional, we will have to resort to unsubstantiated assumptions: mine is that, in this case, two moves forward would be ideal since that is the time necessary to get to the farthest point from any given starting position. The reward function also holds potential for alterations. Although untested, I have every faith that a network could be improved by designing a reward function so that it accounts not just for the immediate score received, but also adds or reduces points based on the reachable gates. For example, if a move moves the mover into a gate that doubles its score, but all three reachable gates from that position halve the score, then it the reward for that move should be somewhere around zero. Without such an addition, the network is likely to struggle in outperforming a simple greedy method. The number of games used in training the network could also greatly affect its final performance. The more testing games, the better the network will be, but there is a threshold after which the gains are minimal compared to the complexity and time they require. Given the relative simplicity of the game, the size of the training set should not need to be too massive, but this is something that would really require testing to confirm, which is impossible to perform with a network that does not work. Although the network part of my program did not work, I was able to test the performance of random actions with greedy actions, and the results were, unsurprisingly, wildly in favor of the greedy method. The one time I was able to get the neural network to do something, the only action it ever chose was to remain stationary, performing roughly equivalent to the random action. I was able to determine that this was because the network was not actually training and did nothing at all. I apologize. I know this is a very bad project. I tried my best. Merry Christmas!