Scott Smith

Professor Balogun

CS-370 Current/Emerging Trends in CS

10/8/2021

**Design Defense**

Solving a maze successfully requires critical thinking and awareness. As humans, we are often unaware of these problem-solving processes taking place as we solve simple games such as Treasure Hunt Game. Further, what goes into solving a problem domain such as this in the fastest and most rewarding way possible? First, we analyze the problem domain (the maze) as explicitly as possible noting the moves we can and cannot do, such as moving from space to space, exiting the game board, or stepping on a dark space. In seconds, most of us can doodle a line from free space to free space indicating a successful path to finding the treasure at the end of the puzzle. We may glance over the maze for a few more moments and simplify the path we have taken to minimize the number of moves required and decrease the amount of time it takes to reach this point. We have defined an optimum policy towards solving the puzzle.

For our puzzle game, however, we’d like to implement an intelligent agent using artificial intelligence or more precisely, a reinforcement learning algorithm, that will allow a CPU controlled agent to solve puzzles and compete against human players. To accomplish this task, we have chosen to use a Deep Q-Learning network. Deep Q-Learning is based on Q-Learning, a problem-solving algorithm that involves initializing Q-tables with Q values, or levels of quality of unique input states, by carrying out every possible action in all unique states of an environment, noting the rewards, and ultimately finding an optimal solution policy based on the sum of these Q values (Gulli & Pal 2017). This sure sounds like a lot of work even for a simple scenario like Treasure Hunt Game. Instead, Deep Q-Learning uses neural networks to approximate these Q-values for all actions that can be taken at the present state (Karagiannakos, 2018). “The biggest [Q-value] output is our next action” (Karagiannakos, 2018). We attach unique action-Q value pairs, based on our collected experience and observations, to unique input states (Wang, 2020). In other words, our intelligent agent engages in random trajectories within the game environment and approximates and records an estimated overall value for the actions it takes from these different positions. Each iteration, the algorithm has more information to develop a better policy towards solving the problem domain. Eventually, its estimations will be optimized, and the agent exploits this solution policy.

As humans, when we solve simple problems like this, we don’t necessarily attach Q-value approximations and take notes of our experience before developing the optimum solution. Our approach seems more intuitive, but this may be because the analysis we perform is significantly internalized. However, is this not precisely the way we conquer more complex problems, such as developing a highly complex software application? For larger problems, we do collect information from our experiences and engage with that experience to make better decisions. Whether we realize it, both human minds and intelligent agents eventually converge upon optimal results in the end based on a reward system. These parameters need to be defined more precisely for a machine to understand the goal of undertaking such a task, whereas humans can be more easily incentivized by simply understanding that the game is won when we reach the treasure. While other approaches exist for creating intelligent agents that can solve challenging problem domains such as policy based RL algorithms, the idea of providing incentives to make proper decisions remains the key motivating force for human and machine learning alike.

A challenge in developing an agent based on Deep Q-Learning is avoiding local maxima and focusing the agent on solving problems globally (Gulli & Pal, 2017). If we are randomizing the agent’s trajectories each episode, is there any guarantee the agent won’t fall into the trap of performing the same actions repeatedly? In a domain where performance demands are highly critical, we need to implement ways to mitigate these issues. We define what is known as an exploration factor (epsilon) which is a level of randomness in which the agent will forget about what it already knows (exploitation) and engage in a random and valid action from its current position. Through many trials, we found an exploration factor of 0.1 (10%) produced optimal levels of performance and motivated our agent to solve its problem domain after only 121 epochs. Both higher and lower values of exploration resulted in poorer performance by delaying the upward momentum of the agent’s win rate therefore increasing the time and number of epochs required to solve the puzzle. The overarching purpose of this process is to approximate the most rewarding actions quickly and efficiently from all possible free spaces within the game environment. Using principles of exploration and exploitation in this proportion allowed our agent to remember its learned experiences, discover new opportunities, and record these events as experience replays which are ultimately used to determine an optimal policy.

As mentioned previously, intelligent agents developed using Deep Q-Learning algorithms must engage in a series of epochs (iterations) before solving complex problems. Prior to training, we create a neural network model consisting of a linear stack of layers which pass through two fully connected dense layers the size of the maze and generate an output that will be one of all possible actions. Naturally, the first step is to engage the agent in a loop since it will be repeating similar processes over many epochs before converging onto a solution. The game environment has been predefined, so at the beginning of each epoch the agent selects a random free cell within the maze before taking any action. We reset the maze with the agent at this position and return to it the current environment state. Before taking any actions, we reset the number of episodes to zero so we can be aware of the number of actions taken by the agent during each epoch upon winning or losing. We also zero out a loss value to be used later in our model evaluation. We now engage the agent within the environment. The action taken by the agent is chosen based on our exploration factor and will either be a random and valid action (exploration), or an action based on the agent’s experience (exploitation). The action is taken and all gameplay data including the previous game state, the action taken, the reward received from the environment, the new game state, as well as the current game status (win, lose, not over) is stored as an experience replay. We fit and pass the training data to train our model and evaluate our current loss. If at this point, the game status is either ‘win’ or ‘lose’, we append our win history and calculate our win rate. The goal is for our agent to learn and approximate Q values for all free cells, in all cases solve the maze, and achieve a win rate above the threshold. We perform a final check verifying the agent has developed the necessary knowledge to successfully solve the maze with an optimal policy, and if this passes, training ends. Finally, the agent is reset to the starting position of the maze before engaging in the game loop. The results are drawn to the screen to display the agent’s solution.

Sources:

Gulli, A., & Pal, S. (2017). *Deep Learning with Keras*. ProQuest Ebook Central

https://ebookcentral-proquest-com.ezproxy.snhu.edu

Karagiannakos, S. (2018, September 30). *Q Learning and Deep Q Networks*. Medium. Retrieved October 1, 2021, from <https://towardsdatascience.com/q-learning-and-deep-q-networks-436380e8396a>.

Wang, M. (2020, November 17). *Deep Q-Learning Tutorial: MinDQN*. Medium. Retrieved September 25, 2021, from https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.