Multi-Agent Reinforcement Learning Using Deep Q-Learning Hide And Seek

Arsham Mehrani, Alondra Sanchez

Abstract

With the growing popularity of reinforcement learning in areas such as game development new frontiers have opened in the world of AI. An example of such work is the Open AI Hide-and-Seek project. This team set the goal to implement artificial intelligence to this game to the best of our abilities. In our attempt, we designed and built the game from ground up using the Python and Pygame libraries. Much like the open AI [Baker *et al.*, 2019] implementation of this game, the team implemented a ray-casting algorithm.

We created the reinforcement learning agent by taking a Deep Q-Learning approach. Additionally, we designed and implemented a neural network to predict the Q-values using a player's state. The Q-values were then mapped to the actions possible for the agent at every given moment. The agent will then use epsilon greedy method to take the best action possible while maintaining a degree of exploration. Overtime the epsilon will decrease as the agent becomes more experienced with the environment. It is important to note that neither of the two agents have a broad knowledge of the map, and must learn this by the use of ray-casting and exploration.

Introduction

This team is implementing the concept known as Q-learning in the field of Artificial Intelligence to the classic childhood game Hide-and-Seek. The graphics for the game are handled through the Pygame platform. The game consists of a maze and two agents - the hider and seeker. The hider's objective is to find the flag in the maze, capture it, and make its way to the exit without being captured by the seeker. The seeker's objective is to capture the hider before the seeker can successfully capture the flag and make its way to the exit. The players and flag always start at the same position. Both agents are trained to detect their surroundings through the use of a ray casting algorithm and a neural network. A sequential neural network was created to train agents on which actions (move up, move down, turn left, and turn right, no movement) it should take to win the game as quickly as possible. This densely connected neural network implements a deep Q-learning algorithm to compute calculations. Hundreds of episodes were played to test scenarios and measure an agent's learning improvement.

Related Work

Open AI has a similar project, which is one of our main inspirations for this project. The work of Open AI is listed as the first citation in the references. Open AI's implementation of this game is however different from ours in many factors. Open AI uses AI players for both the hiders and the seekers. They also use move-able obstacles as tools for the hiders to hide behind or use against the seeker. Additionally, the objectives in Open AI's implementation are different; the main objective for the hiders is to stay hidden while the seeker's objective is to keep the hiders in the line of sight. Lastly, Open AI's implementation has no time control or a definitive end. This is different from our project in that, our project has a definitive winner and loser every round. Our version of hide-and-seek game is more structured having clear tasks for each player to accomplish in order to win. There is also an ultimate best answer for the AI seeker. Our game will feature walls throughout the playing field to challenge the agents in navigating the field to pose a challenge to "throw off" our AI. Both players have a constrained field of vision. Neither agent will have advantage compared to the other with this applied field of vision. This way we train the AI aggressively to improve and find the best approach in attempting to win the game.

Method

The Team has created the game of hide and seek with two AI agents using reinforcement learning. Before creating the AI agents and starting the training process, the team built a game from ground up using python and pygame library. Using object-oriented programming the team managed to create a game modular enough to be able to adapt to any sort of AI learning. Everything from the map/maze to hider and seeker's vision is dynamically designed to be adjustable based on user needs.

Algorithms

With the nondeterministic nature of this game and the repetitive nature of the learning process, we have determined that

the best course of action with our current information is reinforcement learning and more specifically deep Q-learning. We use deep Q-learning instead of traditional Q learning because of the vast domain of the states possible for our agents. The agents use the epsilon greedy method to explore and exploit the environment and as a result learn the tasks they ought to execute.

Ray Casting and Player Vision

Like an autonomous car using LIDAR technology to detect the surrounding, each agent in this game uses our ray-casting method to become aware of their surroundings. Both players project 120 rays up to a maximum distance of 500 pixels in a 60° field of vision simulating a sense of vision. Each ray is casted until its maximum depth has been met or until the rays hit an object including walls, other players, and flags.

At each frame of the game a for loop iterates through all the rays and extends each ray until the maximum depth is met or until an object is detected. We use this algorithm to detect walls, flags, and other players. The formula used to calculate the tip of every given ray at each point is provided below:

$$target(x) = player(x) - sin(\theta) * depth \quad (1)$$

$$target(y) = player(y) - cos(\theta) * depth \quad (2)$$

The players direction, 0°, starts directly to the right of each player but contrary to a normal unit circle, the angle progression is clockwise due to the board being reflected over the X-axis (with 90° being directly below). At every position we use the angle and the current depth of the rays casted to calculate the delta-x for the tip of every ray with respect to the player.

If a ray collides with an object (the target x,y coordinates match a particular object or another player) that object becomes visible to the player. The depth of the current ray is recorded as the distance directly to the object to be used for training. The player effectively uses this mechanism to view its surroundings.

Deep Q Learning

As the agent moves through the environment and takes action, it learns to corrolate a particular state (x, y, θ , distance to the walls) to an action. The agent will then choose an action (left, up, right, down, no movement) based on the Q-values calculated by the Neural Network. Q-values are essentially the highest reward that the agent predicts to receive as a result of taking a particular action. At every state (x, y, θ, distance to the walls) the agent will calculate the Q-value for every action (left, up, right, down, no movement) and choose the highest/best action to take. The Q-Learning Agent learns to play the game based on an "Off-Policy" method. This means that it takes actions that will maximize the future rewards, and the Q-values are updated assuming that the best action was chosen, even if it was not.

$$Q_{new}(S_t, a_t) = Q(S, a) + \alpha * (r_t + \gamma * Q_{MAX}(S_{t+1}, a) - Q(S_t - a_t)) \quad (3)$$

Epsilon-Greedy Decay

To maximize the Q-values and teach the game to a novice agent, the agent must first have a degree of random exploration. Using some random exploration, the agent gains intuition about its surroundings and the best actions given each state. This is achieved using epsilon-greedy coupled with epsilon decay method. The agent will take random actions with the probability of ϵ, and use its prior knowledge with the probability of 1-ϵ, overtime the epsilon is decreased as the agent becomes more experienced with the environment.

Neural Network

The architecture of the neural network is shown in fig.3. The Neural Network consists of 2 hidden layers of 8 densely connected neurons. An input layer with 4 densely connected neurons, and an output layer with 5 densely connected neurons. All these layers use the 'ReLU' activation function. At every state the neural network takes in 4 values: 1. X coordinate, 2. Y coordinate, 3. Angle of the player, and 4. The distance from the player to nearest walls. This information is used to calculate the Q-value for each action: 1. Turn left, 2. Move forward, 3. Turn right, 4. Move back, 5. Do nothing.

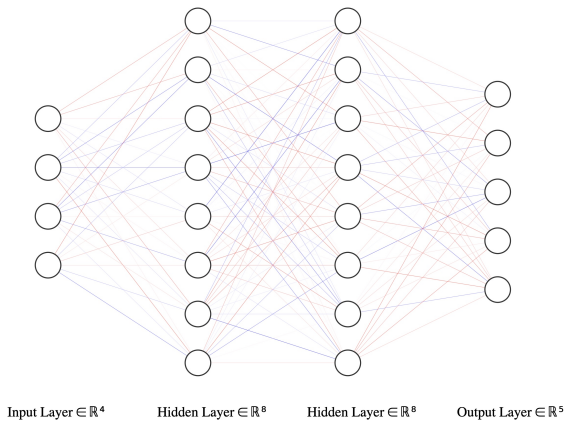

Figure 1: The Neural Network visualized

Game Dynamics

Board

The board is constructed of 100 squares of 80x80 pixels. The wall segments are always a full square space. The board is constructed dynamically and can change based on a new input pattern. Every part of the game is dynamically designed to consider changes in the shape of the board. The player and the seeker AI agents will each start on opposite

sides of the board. The hider begins on the bottom right corner, whereas the seeker's starting position is on the upper left corner. Each agent has different objectives. The finish square appears once the hider successfully captured the flag before being captured by the seeker. In the event that the game times out, or either player wins the map is restored back to its original shape (no exit) and the players are returned to their starting positions. The flag is also reset to appear on the upper right corner of the board.

Hider

The goal for the player is to first capture the flag then proceed to the finish tile. The agent needs to learn how to navigate the maze while avoiding bumping into walls and running in the seekers field of view. This agent can observe 60 field of view set to the direction it is facing. Meaning, they can see 30 to the front-left and 30 to the front-right. The player has a limited time (20 seconds) to clear the map, this restriction is in place to prevent the user from sitting idle.

Seeker

The goal of the seeker is to locate the hider. Using its field of vision, the seeker can locate and capture the hider. After many episodes, the AI should be able to recognize that the best course of action for it is to catch the player at the finish square. Because the hider needs to clear the maze and go to the finish square within a limited amount of time the AI could just wait at the finish square and block it.

Results

The results expected after running our program for a couple hours was to ultimately have the hider agent learn which actions to take in order to make its way to the flag and back to the exit without running into the seeker in the most efficient way as possible. Though we were not able to test the how well the hider agent did when it came to returning back to the exit and win, we managed to see major improvement in its learning in the way it learned to capture the flag. Overall, all the trial runs we tested resulted in the successful training of an agent. To test the learning of the hider agent we attempted different trials with slight modifications to the neural network and the agent's environment in every episode.

Initial Attempt

The original neural network that was used successfully trained the hider to take actions that would lead it to capture the flag as well as avoid bumping into walls. During the early episodes of the game, the hider's movement from its starting position was minimal. Rather than moving towards the flag, the seeker remained in relatively the same location and turned in circles. After 2 hours (about 720 episodes) the hiders improvement was just what we had expected; it learned what direction and actions to take to capture the flag as soon as the game began. Based on these results, our team decided to attempt to test whether the hider could also learn to interact with the seeker and avoid the agent.

Second Attempt

After analyzing the results from our initial run of the program, we attempted to increase the speed of our results. We decided to shorten the amount of times the game was rendered in Pygame and removed drawing the rays that were drawn. We also took into account the information being inputted to the neural network. To increase the learning progression of the hider, we added two additional inputs to the network - the distance the hider has to the seeker when it was within close proximity, and the distance from the hider's location to the flag. This resulted in similar results to the initial run, however we were able to observe the improvement of the agent in less time.

Improvements

In order to improve the accuracy of the neural network compared to our initial design for the network we added two more parameters: 1. Distance to flag, 2. Distance to seeker. These parameters combined with the four previously mentioned parameters improved the training time and accuracy of the agent's decisions. It is crucial to note that the newly added parameters only apply if the object is within the field of view; otherwise, the input is infinity. These improvements resulted in interesting phenomena where the hider managed to dodge the seeker for brief time. We also noticed that by expanding the hider's field of vision, it could see the flag from a further distance and eventually get to it faster. These improvements resulted in faster training time, and more visible progress through the run of episodes. Through 700 episodes, the hider seemed to be able to capture the flag consistently.

Final Attempt

After adjusting the inputs of our neural network, and seeing rapid improvements to our agents our team decided to challenge the hider with more obstacles. Rather than allowing the hider to freely navigate the field, we manually took control over the hider to move it so that it captured the hider before the hider could make it to the flag. After approximately an hour of repeating this pattern, the hider learned to dodge the location where it was being captured. Once again, these results were exactly what we expected and wanted to happen. Overall the attempts made to train the hider agent were successful. The agent managed to capture the flag and learned to recognize how to avoid the seeker during its path to the flag.

Future Work

This project was more time-consuming and required much more training time than we had originally anticipated. Given more time, we plan to include the following improvements to the future generations of this model:

- integrate the seeker AI agent.
- Dedicate more time to training.
- Improving the code algorithms.
- Adjust policy rewards and penalties Fine tune the parameters fed into the model.

Seeker AI agent

Our game currently has one AI agent- the hider. Given more time, we would implement a similar learning approach for the seeker. The most significant difference from this agent compared to the hider would be that it might not need to know the same environmental details the hider knows. We would like for the seeker to be flagged when the hider captures the flag so hint the hider is more than likely already making its way back to the exit tile. With these adjustments it would be interesting to observe how it impacts the hiders learning performance.

Training

Given our time constraints, our team was not able to see the learning performance of the hider past the flag being captured. In the future we would come back and allow for the learning to be done in a matter of hours/days compared to the two hours in order to gain a greater understanding on how the agent is performing overtime. We would expect our results to be similar to the ones we had in our initial runs of the game possibly resulting in the game actually being won by one of the two players assuming the seeker is also an AI agent.

Policy

The policy we currently have integrated in our game works;however, given the chance we improve this game, we would adjust the policy to add new rewards and penalties. We would consider reward the seeker agent for being near the hider and penalize the seeker. We would also consider rewarding the seeker for escaping (moving away from the hider) being in the field of range near a hider.

Improving Algorithms

Assuming our team would be implementing the second AI agent, we would revisit the current algorithms such as raycasting, to make improvements to the way the agents learn about their environment and progress in the game.

Model Inputs

Though the current model we designed train the agents has proven it can successfully provide way way for the hider agent to learn, we would consider making changes to the inputs it handles. When we jumped from feeding the model four inputs to six inputs, we were able to observe a significant improvement to how the hider agent learned. Adding another parameter or two has the potential to help improve our results.

Conclusion

The project itself was a learning curve for both partners involved. As partners we accomplished creating a solid game foundation with proper functionality between players and their environment. Another accomplishment our team had was designing and creating a sequential network capable of receiving an agent's state as input to output the estimated q-values from actions taken at that state - a calculation vital to an agent's learning process. Additionally, through trial and error, the team formulated a policy to reward and penalize an agent for the actions they made. By completing the task of creating these foundational elements, the end result of our game was seeing the hider agent slowly improve its performance in navigating its way to the flag in the beginning of each episode. Due to time constraints we unfortunately were not able to implement an AI agent for the hider. In the chance we revisit this project in the future, a task we would set to accomplish would be creating an additional agent - the hider. In order to increase learning progression for both agents, we would make adjustments to our policy as well as alter the inputs given to the neural network by adding more attributes to an agent's state. Overall, this team is walking away from this project with a much greater understanding and appreciation for Deep Q-learning and neural networks.

References

Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *CoRR*, abs/1909.07528, 2019.

J. Chapman and M. Lechner, "Keras Documentation: Deep Q-learning for atari breakout," Keras, 23-May-2020. [Online]. Available: https://keras.io/examples/rl/deepqnetworkbreakout/. [Accessed: 15-Apr-2022].

Mordatch, Igor and P. Abbeel. "Emergence of Grounded Compositional Language in Multi-Agent Populations." AAAI (2018).