AI HOOPS - A Basketball Simulation

Ethan Carpenter, Anish Sahoo and Sana Ali
Summer 2024
CS 4100 - Artificial Intelligence
Northeastern University,
Boston, MA
{carpenter.et, sahoo.an, ali.sa}@northeastern.edu
Github Repository: github.com/anish-sahoo/AI-Hoops

***Abstract*— For our final project, we trained an AI to play the Atari game simulation Double Dunk using a custom-built Double Deep Q-Network. We processed the RAM data to determine the best actions to minimize the time to end a game. Our project demonstrates the potential of utilizing RAM-based reinforcement learning in sports simulation, a solution that has yet to be widely attempted. This report offers insights into the strategic decision-making processes of our AI. It opens new avenues for applying artificial intelligence in complex multi-agent settings and more scalable AI solutions across various domains.**

I. INTRODUCTION

The traditional approaches in the Atari Learning Environment are tailored to first-person navigational games, which leverage visual data to train one or multiple agents. While these games are classic, they offer limited complexity regarding strategic depth and multi-agent dynamics. The core problem we aim to address is to implement an intelligent agent capable of playing a simulated variant of basketball while optimizing the finishing time. Atari Double Dunk is a simplified 1-hoop basketball implementation with two players on each team. Interestingly, however, both players in a team are controlled by one joystick. While we could not curate an environment with true multi-agent learning within the limited time frame, we picked Double Dunk as a good substitute.

Our group has chosen to utilize the game's RAM data instead of image pixel data. This approach is less common and potentially much more efficient since using the RAM has 128 inputs while using color or gray-scale images has 100800 or 33600 inputs, respectively. Reducing model size is essential because it allows for the AI to be added to a broader array of products such as embedded devices with limited computational and memory resources. The decision to use RAM also means that the model is significantly smaller and does not require convolutional layers.

This challenges the agent to learn and react to a dynamic environment where decisions are predictive and strategic. By training the agent with Double Dunk, we aim to explore deeper aspects of game theory and decision-making processes using reinforcement learning.

These constraints enhance our simulated basketball environment and contribute to the broader field of artificial intelligence by providing insights into how agents can optimize strategies in rule-based systems. Traditional reinforcement learning is limited to low-dimensional input. By employing RAM data over pixel image data to solve our problem, our approach reduces the computational complexity with high-dimensional input spaces. This makes training more efficient and aligns with the growing demand for scalable AI solutions in various domains.

II. RELATED WORKS

Our initial exploration found that our problem had previously been solved using pixel image data. One of our ideas was to use a Deep Q Network (DQN) Learning algorithm to solve our problem.A paper by Mnih et al. employed a DQN model to master a variety of Atari 2600 games using pixel image data as input. The results of this experiment indicated that Double Dunk could not achieve scores better than the average human. This was attributed to the tendency of DQN models to overestimate Q-values of potential actions in a given game state.

After tweaking our approach, we adjusted our strategy and ultimately decided to try implementing a Double Deep Q Network along with RAM data. As mentioned by Bellemare et al., DDQNs can overcome the overestimation seen with standard DQN models by utilizing a dual architecture—using a fixed Q target to stabilize the learning process and selecting the best action. This approach aims to overcome low convergence and achieve efficient processing.

Additionally, an experiment by Shangtong et al. demonstrated that the replay buffer can drastically reduce the number of episodes required to train, and the size of the replay buffer can significantly affect model performance. The critical idea of experience replay is to train the agent with the transitions sampled from the buffer of previously experienced transitions. While increasing the buffer size can

potentially increase the time necessary for the algorithm to execute, this does not guarantee optimized performance. We thought it would be interesting to modify the size of the buffer from 1000000 to identify an optimal size that balances training speed with learning effectiveness, as many papers published within this domain do not attempt this.

III. METHODS

A. *Offline Learning*

During the initial phase of this project, we attempted to use offline learning by collecting our data to train our AI. This approach would have allowed us to tweak the input as the AI adapted to the environment and mitigate the risk of over-fitting the data. We aimed to structure the training data to simulate the dynamics of a real-life basketball game. However, after further analysis, this method of training showed little change in the policy. Replicating the interactions and constraints of the game by generating our data would be highly resource-intensive and did not fully reflect the level of complexity we wanted. Additionally, storing and processing our data would have impacted the overall speed and efficiency of the training process and could skew our results and model performance. After training the agent for a thousand episodes, we changed our approach, as outlined in the next section.

B. *Online Learning*

To address the instability in training and the flaws of our initial approach with offline learning, we ultimately decided to implement epsilon-greedy-based action selection to overcome the issue of unstable training. We replaced our *DataLoader* with a *ReplayBuffer* to store past observations for future sampling. This approach has demonstrated to be slowly 'learning,' as we saw some primitive, clever plays by the AI during the state of the game. However, while this approach produced substantial results, more fine-tuning is necessary to get the AI to learn strategy and environment patterns better.

C. *Double Deep Q-Learning Network*

Given the issues referenced, we shifted to using the Atari Double Dunk environment, which allows us to leverage 128 bytes of RAM data to store the game data. We also built a DDQN model from scratch to train our agent. We use the following formula to form a new estimate of the Q value for some state/action pair $Q(S_t, A_t)$:

$$Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbf{max}_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Along with this, we use Q-loss as the loss function:

$$Q \textbf{ Loss} = (\textbf{Target } Q \textbf{ Value} - \textbf{Predicted } Q \textbf{ Value})^2$$

The DDQN model was designed using PyTorch to tailor input from the state of the game and output actions the agent can take in the Double Dunk environment. As shown in Figure 1, each state representation the network receives as inputs consists of 128 feature nodes, representing the state of the vector from the game's RAM (128 bytes). This data consists of game statistics and metrics such as player position on the court and ball status. The first and second sequential linear layer outputs 128 and 64 dimensions and is followed by a ReLU activation function, which introduces non-linearity to the model. This allows the model to learn more complex patterns and identify the patterns and interactions with the environment throughout the state of the game.

The third and fourth sequential linear layers, with 64 and 32 nodes, respectively, help the model focus on the features relevant to the task. It further defines the learning and strategic decision-making process for the AI. The output dimension is reduced in the following layers with a subsequent ReLU activation function. The final output layer converts the 32-dimensional representation to the number of possible actions. The 18 nodes correspond to the variety of potential actions the agent can choose during the game state. Double Dunk uses the entire action space provided by gym-like environments for 18 possible actions. At each layer, the ReLU activation is crucial to ensure that the model is dynamically learning based on the environment and strategically making decisions, not just passively passing input to the model.

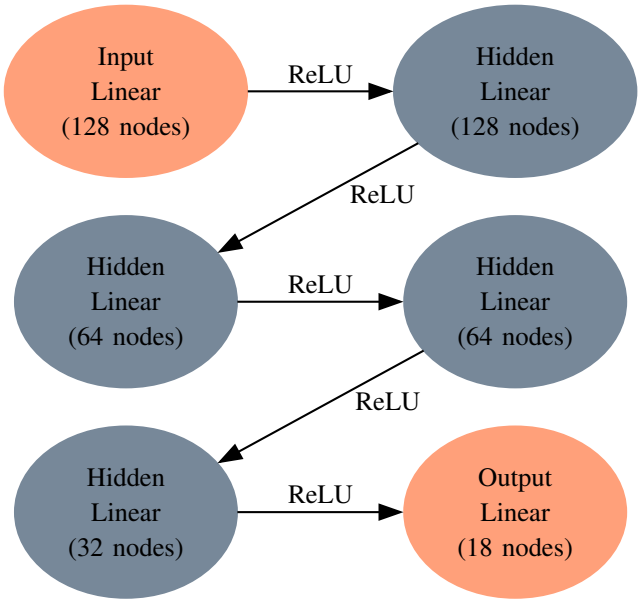

Fig. 1: Neural Net Layout

Throughout the game environment, the network uses a strategic decision-making process to decide the best action to take next based on the current state of the Double Dunk environment. The decision-making is facilitated by feeding

the game's current state through the model to obtain Q-values for all possible potential actions and then selecting the action with the highest resulting Q-value. This occurs either directly or after a policy decision, specifically epsilon-greedy, to balance exploration with exploitation. This network setup allows the DDQN to refine focus toward strategic decision-making based on the game dynamics encoded in the RAM data. This will, in turn, make the agent's training process more robust and faster.

IV. OBSERVATIONS

The total training time allocated for this project took 20 hours combined with several different combinations of hyper-parameters! The plot in Figure 2 illustrates the model's training dynamics during a training run with over 500 episodes, showing both loss and cumulative rewards as the network learns and adapts. The progress across these episodes shows how the agent's performance changes as it gains more experience through repeated play.

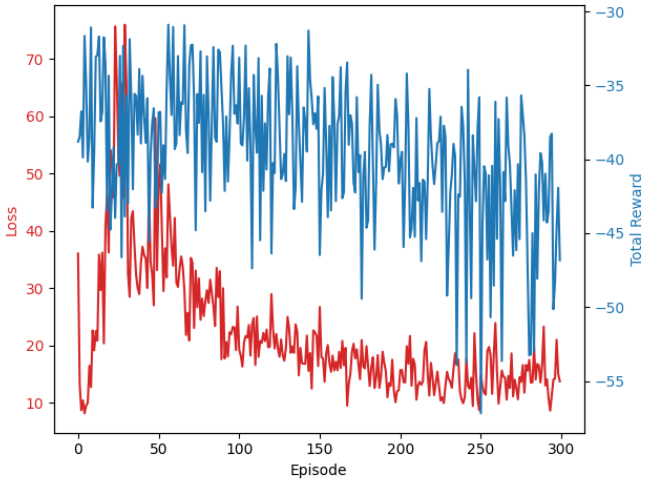

Fig. 2: Metrics from a training run

Initially, we used extremely large batch sizes and about 100 episodes. The low number of epochs was partly due to the large batch size and the relative lack of change in the episodes following the 100_{th} episode. After we tested these hyper-parameters, we tested using a batch size of 64 and 300 episodes. In this case, the reward was becoming more hostile, and the rate of change loss function was decreasing. Due to this, we stopped at 300 episodes. The industry expert Raj Venkat recommended increasing the number of episodes to over 1000. We ran a few training runs with 1,000 and 2,000 episodes, but it appears to need orders of magnitude more episodes.

By visualizing and analyzing how the model interacts with the state of the game environment, we discovered some of the strategies that the AI learned and developed. In the context of the game, two players are on each team in Double Dunk: Mr. Inside and Mr. Outside. The bigger player (Mr.Inside) is great at rebounding and blocking shots, can dunk his shots easier, and sets picks for Mr.Outside. The smaller player (Mr. Outside) is adept at stealing the ball and is great at making long shots.

When Mr. Outside had possession of the ball during the state of the game, the AI would take the shot. When Mr. Inside had the ball, the AI would sometimes choose to dunk it, generally from the side of the net, instead of directly dunking on the net. Additionally, Mr. Outside seemed to prefer taking long shots from the edge of the defined boundaries of the court.

Throughout the episode progression in Figure 2, we observe that the loss initially goes up before the first 50 episodes and gradually starts to fall. When we increase our batch size, the time taken for the loss to reach a maximum and start falling elapsed. As our goal for this project was to optimize the finishing time of the game as an alternative to rewards, the reward per episode is highly variable. We attempted to train the AI with a small negative reward at each time step and no negative reward. The purpose was to see if there was a noticeable difference between them for runs with less than or equal to 1,000 episodes. We could not see any noticeable difference.

We faced problems with using the RAM state as input for our network: it was difficult to modify rewards to our liking and tailor the environment, as we needed access to what the state actually looked like. Due to hardware limitations and time constraints for this project, we could run it for fewer episodes than we originally planned and wanted. Several articles referenced throughout this project's duration mention training over 100,000 episodes. Our simulation currently only runs up to 1000 episodes, so we have much room for improvement in training our agent!

V. DISCUSSION

In the future, our group would like to further our research by looking into an approach combining offline and online learning and using generated data for the replay buffer and epsilon greedy action selection. We may also try some form of parallel learning where we train multiple neural network models at the same time and generate our final model using the metric of their weights. These improvements may enhance our model entirely and speed up the training time. To enhance the model's performance and stability, several other improvements can be considered:

1) Varying the hyper-parameters could allow for better accuracy and training time. To do this we could use a search such as random search.

2) Increasing the training time and decreasing the epsilon decay rate would allow for the AI to be more stable in rare situations.

3) Using software such as *Lime* would allow for visualization of the impact of each cell in the ram on the model. Ignoring these would enable the model to ignore input that does not significantly affect the output, similar to adding dropout to the model. The advantage of this over dropout would be that it is deterministic, applied from the start, and could be added with dropout.

VI. FUTURE WORK

Going forward, we want to optimize this project by using multiple AIs to control both teams in Double Dunk. This would give better metrics on how good the AI is at the game compared to other AIs since the number of games won when playing against other AIs yields a quantitative method of ranking. This would also allow the AI to play against itself, which would allow the AI to focus on learning about situations that it would realistically encounter often.

Additionally, we would like to test our architecture's performance on the other Atari games. The input and output layers are not a hard-coded size, which allows the model architecture to be applied to other games without having to change any code, even if they do not use the full RAM or action space.

VII. TEAM CONTRIBUTIONS

All members worked on the code documentation, final presentation, and report. Anish implemented a DDQN model from scratch. Ethan implemented the data collection script and the multi-agent games. Sana researched and reviewed our developing implementations and compiled the documentation, report, and slides.

VIII. REFERENCES

- Luo, Yudong, Oliver Schulte, and Pascal Poupart. "Inverse reinforcement learning for team sports: Valuing actions and players." IJCAI. 2020.

- L. Antão, A. Sousa, L. P. Reis and G. Gonçalves, "Learning to Play Precision Ball Sports from scratch: a Deep Reinforcement Learning Approach," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-8.

- Jia, Hangtian, et al. "Fever basketball: A complex, flexible, and asynchronized sports game environment for multi-agent reinforcement learning." arXiv preprint arXiv:2012.03204 (2020).

- Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In International Conference on Machine Learning, pp. 449–458. PMLR, 2017.
- Human-Level Control through Deep Reinforcement Learning,storage.googleapis.com/deepmind-media/DDQN/DDQNNaturePaper.pdf.

- "Atari Double Dunk Environment." Endtoend.Ai, https://www.endtoend.ai/envs/gym/atari/double-dunk/.

- "Deep Q-Learning with Keras and Gym." Deep Q-Learning with Keras and Gym Keon's Blog, keon.github.io/deep-q-learning/.

- Shangtong Zhang, et al. "How Large Should the Replay Buffer Be?" Artificial Intelligence Stack Exchange, 1 Sept. 1964, https://arxiv.org/pdf/1712.01275

- , Thibault, et al. Large Batch Experience Replay, proceedings.mlr.press/v162/lahire22a/lahire22a.pdf. https://proceedings.mlr.press/v162/lahire22a/lahire22a.pdf

- Feng, Dennis. "Learnings from Reproducing DQN for Atari Games." Medium, Towards Data Science, 11 Aug. 2021, towardsdatascience.com/learnings-from-reproducing-dqn-for-atari-games-1630d35f01a911cf.

- Hui, Jonathan. "RL-Tips on Reinforcement Learning." Medium, 6 Feb. 2023, jonathan-hui.medium.com/rl-tips-on-reinforcement-learning-fbd121111775. Accessed 15 Aug. 2024.

- "Atariage." AtariAge News RSS, https://atariage.com/manual_html_page.php?SoftwareLabelID=153