Behavior-Oriented Neural Networks: Developing an Imitation Learning Agent

Brayden DeWitt
University of Texas at Austin
Austin, TX 78712

Abstract

This paper explores the application of deep learning techniques, specifically focusing on imitation learning, to train a SuperTuxKart player agent to score goals and win matches against opponents in ice-hockey. Methods for generating optimal training data, creating a neural network, and improving the model are discussed. Furthermore, an analysis of the developed agent is conducted, proving the agent's success in scoring goals.

1 Introduction

1.1 Project Scope and Goals

SuperTuxKart is an open-source arcade with numerous game modes such as racing, capture-the-flag, and soccer/ice-hockey [1]. The goal of this project is to use deep learning techniques to create a deep neural network that will teach an ice-hockey player to score goals and defeat opponents in matches.

This project fits nicely into current deep learning research. While some focus with deep learning is on generative modeling, such as large language models (LLMs) and natural language processing (NLP), there is other research done that focuses on video games and artificial intelligence. Some examples are computer-bots that are designed to help new players learn a videogame, such as Fortnite. These bots are trained to compete against a new player but perform at a very low level – allowing the player to learn basic controls and actions before being matched against real, human players [2]. Similarly, in kart racing games such as Mario Kart, a player needs several computer opponents to partake in the same race to create competition. These opponents are trained on in-game data to learn controls and perform at the appropriate skill level. This project falls into the same research area, as the end goal is for our agent to successfully learn to score goals against opponents.

1.2 Motivation for Work

With the focus on developing an agent to compete successfully in a video-game environment, our interests led us to focus on a state-based agent. With access to several data points at every state in a game, we can train an agent to learn from other, previously-developed agents that perform well – also known as imitation learning. This allows for a supervised neural network that trains on high-performing agents, learning the best actions to take in attempt to score goals. With the correct data, our agent will be able to learn from others, playing in their style and learning when to accelerate, steer, and brake.

2 Methods & Results

2.1 Generating Training Data

To start model development, it is important that good, relevant data is chosen for training.

Within the initial codebase, there are four pre-trained agents available to observe and evaluate. These agents are known as Geoffrey, Jurgen, Yann, and Yoshua. Each agent has a different style of play and performs differently when pitted against one another [3].

After observing all agents' styles of play, Jurgen is the most aggressive and best-performing agent of the four. In terms of imitation learning, Jurgen was determined the best to imitate, since we wanted our agent to consistently score goals.

To generate quality data, 50 matches were played against each agent, including Jurgen playing itself. The puck's starting location and velocity were both randomized, to ensure that our agent could learn from several different initial game conditions. Additionally, the goal-side was changed halfway through, so that our agent could learn to score on the correct goal in the hockey rink – depending on if they were the red team or blue team. Each match was played until the teams scored a combined total of 3 goals, or once 1200 frames were reached – whichever came first.

In total, 200 matches were simulated – creating tens of thousands of states to train on. With imitation learning, however, it is important that the simulated matches are matches where our "supervising" agent – Jurgen – performs well. By manually reviewing videos of the matches, as well as calculating the goals scored per game, the data was filtered to ensure high-performance. If Jurgen did not score in the match, or continually got stuck in a position (requiring rescue), the match file was discarded.

2.2 Feature Extraction

After filtering, there were over 200,000 states of quality data for our agent to learn from. In each state, we needed to extract several features of the game so that our model could learn what to do at different states.

11 features were selected, as shown below in Table 1. These involve features of the kart, the hockey puck, and the goal-line. These features were extracted for each kart on the agent's team – typically two karts.

Table 1: Features for model input.

Group	Feature
Kart	X-coordinate of the kart's center
	Y-coordinate of the kart's center
	Current angle of the kart
Hockey Puck	X-coordinate of the puck's center
	Y-coordinate of the puck's center
	Angle between kart and puck
	Difference in kart angle and kart-puck angle
Goal-line	X-coordinate of the goal-line's center
	Y-coordinate of the goal-line's center
	X-coordinate of the puck to goal-line distance
	Y-coordinate of the puck to goal-line distance

Overall, the goal is to get our agent's karts toward the puck first and score. These 11 selected features help to emphasize this goal. No opponent features were extracted, as those were deemed not as important to achieving our agent's goal.

2.3 Model Structure & Performance

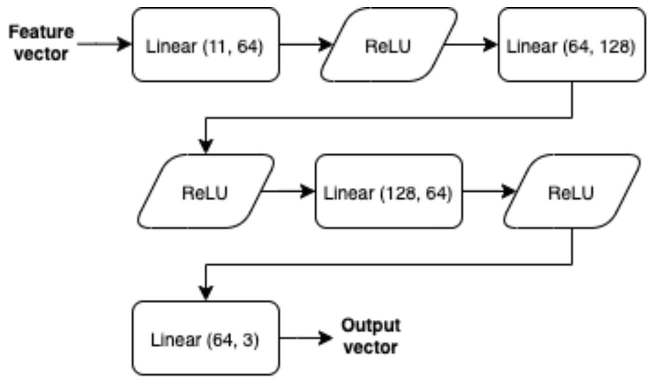
Once data generation and feature extraction were complete, we iterated through different network structures for our agent's training. The goal was to use a deep neural network, but we initially did not know how deep this network needed to be to correctly learn Jurgen's behavior.

2.3.1 First iteration

Our first iteration of a model consisted of three linear layers, each followed by a ReLU (Rectified Linear Unit) activation, and then a fourth linear layer that served as a classification layer for model output. Our model took an input vector of the 11 features previously mentioned and outputted a 3-dimensional vector. This output vector corresponded to the three actions our agent could take: acceleration, steering, and braking.

The model structure is visualized below in Figure 1.

Figure 1: First iteration of neural network.



For training, the Adam algorithm was used for optimization, along with MSE used to calculate loss. After several attempts of training, this model was learning to imitate Jurgen's behavior, but not at a great rate. For example, after playing eight matches with our agent against each of the four provided agents, our agent scored 23 total goals in the 32 matches. Six were scored against Geoffrey, two against Jurgen, ten against Yann, and five against Yoshua.

While it was clear our agent had learned some of the correct behavior to imitate, we wanted it to consistently score in every game – no matter the opponent.

We then reviewed our entire process – from data generation to model training – to ensure that there were no fundamental issues. This led us to the second iteration of our model.

2.3.2 Second iteration

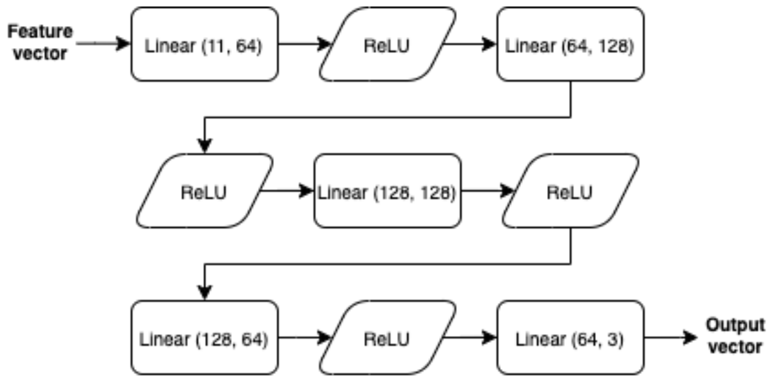
With our second iteration, it was decided that our model should be more complex and deeper. In theory, this would allow our model to learn higher-level features, helping to further our agent towards its goal of imitating Jurgen's behavior and scoring goals.

We added another linear layer, with ReLU activation, at the 128-dimensional space. This allowed the network to learn more about the input features at this level of complexity by capturing more complex relationships.

Additionally, we decided to switch to the AdamW algorithm for optimization during training. AdamW builds upon the Adam optimizer by decoupling the weight decay, treating it as a separate step in optimization. This allows the weight decay to be applied to the weights before the gradient update occurs in training, which has proven to increase model performance [4].

The updated model structure is visualized below in Figure 2.

Figure 2: Second iteration of neural network.



With these changes implemented, we saw the model's performance improve. As we trained, the model continually improved every epoch and scored more goals against the agents. A summary of training and goal statistics are shown below in Table 2. Once again, our agent played eight matches against each of the four agents, for a total of 32 matches.

Table 2: Training results for improved model.

Epoch	Geoffrey Agent	Jurgen Agent	Yann Agent	Yoshua Agent	Total Goals Scored	Average Goals per Game
10	4	2	6	1	13	0.406
100	6	3	1	6	16	0.500
120	6	2	5	7	20	0.625
140	4	1	10	8	23	0.719
160	10	7	8	5	30	0.938

After 160 epochs, the agent scored 30 total goals in 32 matches. Ten were scored against Geoffrey, seven against Jurgen, eight against Yann, and five against Yoshua.

Comparing these results to those of our first model, there was a significant increase in performance. Our first model scored, on average, 0.719 goals per game. Our updated model scored an average of 0.938 goals scored per game. This is a 30.46% increase in performance when judged on average goals per game.

Our overall goal was to have our developed agent imitate Jurgen's behavior and score goals every match. To evaluate the success of this, we compared our agent's performance to Jurgen's performance – using the same criteria as above.

In 32 total matches, Jurgen scored eleven goals against Geoffrey, five against Jurgen (when playing against itself), nine goals against Yann, and eight against Yoshua. This is a total of 33 goals in 32 matches, or an average of 1.031 goals per game.

In comparison to Jurgen, there is a 9.44% difference in performance with our developed agent. For imitation learning, this is a relatively small difference – proving that our agent, through imitation learning, has successfully learned most of Jurgen's behavior for the game's large state space.

3 Conclusions

3.1 Project Success

Overall, our deep neural network proved successful. Our end goal was for our agent to correctly imitate the behavior of another agent – in this case, Jurgen. By learning the behavior of another agent through training on thousands of states and extracting important features of the game, our agent determines the correct actions to take to gain control of the hockey puck and score goals.

Throughout our research, we discovered that adding the additional linear layer with ReLU activation helped our model learn more complex relationships between the state space and features, allowing our agent to score more goals against opponents. Additionally, by using the newer, improved optimization algorithm AdamW, the relevant weights in our network were better adjusted to aid in training – leading to more accurate learning, and a 30% increase in performance from our agent.

Deep learning is a rapidly changing, growing field, with new ideas and research constantly being published. By applying deep learning techniques, specifically imitation learning, we successfully developed a new player agent that could hold its ground against opponents and win matches in SuperTuxKart ice-hockey.

4 References

[1] Gagnon, M., & Clemencon, J.-M. (2024). *Discover SuperTuxKart.* SuperTuxKart. https://supertuxkart.net/Main_Page

[2] Cabal, A. (2023, June 19). *Fortnite Bots Uncovered: Revealing the Truth*. Game Champions. https://www.gamechampions.com/en/blog/does-fortnite-have-bots/

[3] Krähenbühl, P. (2019). *SuperTuxKart Ice Hockey*. Deep Learning. https://www.philkr.net/dl_class/homework/final/

[4] Loshchilov, I., & Hutter, F. (2019). *Decoupled Weight Decay Regularization*. International Conference on Learning Representations, New Orleans.