UT Austin - Deep Learning Spring 2024

Group 42: SuperTuxKart Ice Hockey State Based Agent

Reese Williamson, Sooihk Ro, Abelardo Riojas, Azadeh Khoddami

1 Introduction

In this study, we present a state-based agent designed to master the 2v2 ice hockey game within SuperTuxKart. The primary objective of our approach is to develop policies that consistently outperform other AI opponents. To achieve this goal, we employ a multi-phase strategy:

- **Expert Observation and Imitation Learning:** Initial data was gathered by observing an expert agent provided by the TAs, (Philipp Krähenbühl, 2024) `jurgen_agent`, perform driving drills where the agent drives straight to the puck from various positions on the field without any interference. This straightforward task helps to establish baseline behavior patterns for basic navigation and puck interaction.

- **Iterative Refinement with DAgger:** Following the imitation learning phase, our agent engages in simulations against various AI opponents provided by the course. We utilize the Dataset Aggregation for Learning with Graded Expert Rewards (DAgger) framework to iteratively refine our agent's strategies and reactions in dynamic, adversarial conditions, enhancing its adaptability to diverse gameplay scenarios (Ramesh and Montaño, 2017).

Our research aligns with ongoing efforts in deep learning, particularly in the development of neural network architectures for diverse tasks. Additionally, we explore the concept of transfer learning by evaluating the effectiveness of pre-trained models, such as `jurgen_agent`, in our gaming environment. By integrating principles of reinforcement learning, our model learns decision-making processes through interactions with the game environment and feedback in the form of rewards.

While our study does not introduce groundbreaking advancements in natural language processing or computer vision, it showcases the application of deep learning techniques to address complex challenges within a gaming context, exemplified by SuperTuxKart. The project serves as a practical platform for experimentation with reinforcement learning, optimization techniques, and model architecture design to advance the development of AI systems.

The project was focused on implementing a state-based remote, as opposed to an image-based solution, due to the smaller dimensionality of the input space compared to images. With a state-based solution, we can calculate valuable features from the game's direct world state. This approach was deemed much more reliable by our team.

Our decision to emulate the actions of `jurgen_agent` stems from careful analysis of its win-rate percentages and strategic approaches. Notably, `jurgen_agent` demonstrates consistent success by efficiently navigating to the puck, making it a compelling model for imitation.

Our state space comprises 12 features derived from the states of karts on `team1` and the soccer match, with a crucial addition being `kart_distance_to_puck`. Notably, we refrain from incorporating opponent kart states based on empirical evidence indicating the superior performance of `jurgen_agent` compared to other agents, including `yann_agent`, which utilizes opponent state information.

2 Methods

2.1 Overview

Our methodology revolves around implementing a behavior cloning strategy to train our player agent effectively. Initially, we utilize pure imitation learning, leveraging `jurgen_agent` as an expert oracle to guide our agent's decision-making process. Subsequently, we enhance our agent's training through the integration of the DAgger algorithm (Chen and

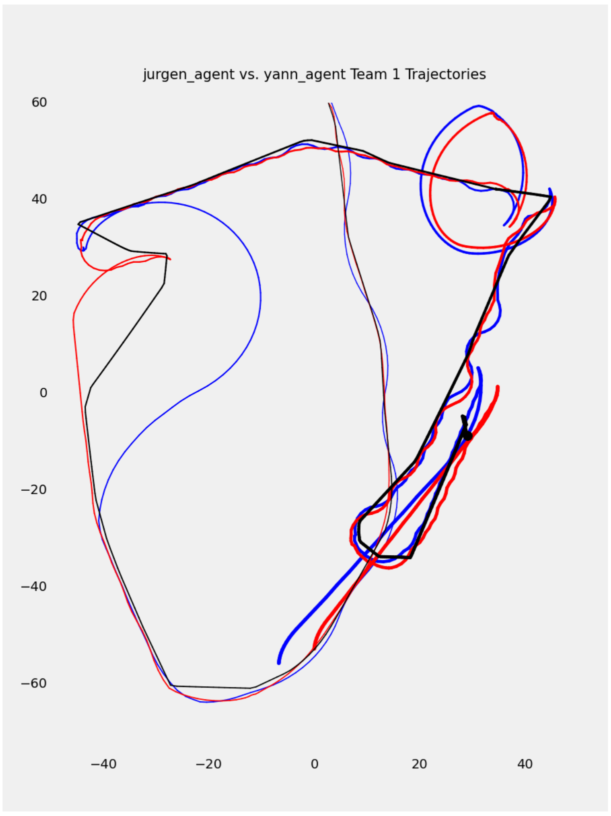

Figure 1: Trajectories of `jurgen_agent` 1 (red) and `jurgen_agent` 2 (blue) as well as the puck (black), with decreasing line widths the further along in the trajectory.

Yi, 2017).

2.2 Gathering Data

Considerable effort was dedicated to simulating the behaviors exhibited by `jurgen_agent` to gather training examples. We created two distinct datasets for this purpose: a drill-based dataset and a tournament-based dataset. The drill-based dataset involved conducting 1v1 matches against an agent programmed to retreat into its own goal (`coward_agent`). Conversely, the tournament-based dataset mirrored the actual gameplay scenarios encountered during evaluation sessions.

The decision to focus on imitating `jurgen_agent`'s capability to locate the puck from any position on the field stemmed from the reproducibility and effectiveness of this behavior in scoring goals. As depicted in Figure 1, trajectories from matches featuring `jurgen_agent` illustrate its adeptness at securing possession of the puck and scoring. From the grader file, `jurgen_agent` scored the highest among all the agents (84/100).

Notably, the inclusion of the additional feature `kart_distance_to_puck` in our input features proved valuable in achieving a better fit for this task. Moreover, it facilitated the truncation of training example frames to instances immediately after the kart successfully interacts with the puck.

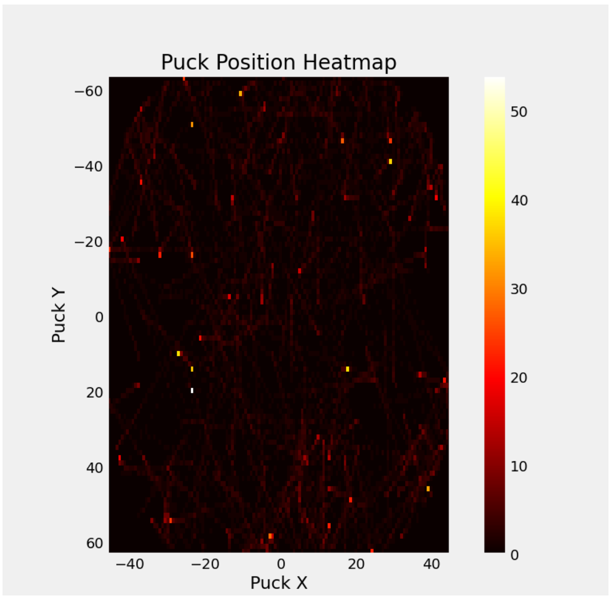

Figure 2: Coverage heatmap of puck positions across all examples in our drill-based dataset. Puck covers most playable areas of the map, although admittedly it is not complete spatial randomness.

In terms of simulation methodology, we opted to randomize the location and velocity of the ball upon match initialization. This choice was motivated by the necessity to ensure comprehensive coverage of the puck across the playing field, crucial for our agent's success. Figure 2 presents the heatmap depicting puck coverage across all training examples in our drill-based dataset.

For the tournament-based dataset, we engaged in 2v2 matches against all state agents provided by the TAs, refraining from randomizing ball locations or velocities to closely replicate the conditions of graded evaluation matches. As this dataset served as the basis for our DAgger training (refer to Training), our objective was twofold: initially teaching our agent to locate the ball, followed by applying this training to excel in 2v2 matches against other agents.

2.3 Architecture

The state-agent model is constructed using a neural network architecture, depicted in Figure 3. The architecture begins with an input layer that receives the game state features, with the dimensionality determined by the number of input features. Subsequently, the model comprises four blocks, where the output of each block serves as the input to the next.

Within each block, two linear layers are employed, followed by Rectified Linear Unit (ReLU) activation functions and dropout layers. These components facilitate linear transformation of input data and introduce non-linearity into the model.

Upon traversing the four blocks, the output features undergo a final sigmoid activation function for each output dimension, ensuring the resultant values are bounded between 0 and 1 before being returned by the `.forward()` method.

Due to the nature of our output actions, (acceleration continuous between 0 and 1, steering continuous between -1 and 1 but only ever -1, 0, or 1 in the data, and braking either 0 or 1) our original plan was to have an output layer for each action and perform different loss calculations on each to form a composite loss.

We experimented with different activation functions and rounding methodologies for each action. For instance, hyperbolic tangent activation was considered for steering, while braking action was thresholded at different values to yield binary outputs. Steering proved particularly challenging as PySuperTuxKart's game engine accepts steering values in the interval [-1, 1] but the actions of `jurgen_agent` were only ever {-1, 0, 1}. This led us to believe that we could treat steering as a classification problem (adding +1 to the steering directions to form class labels) but proved inefficient as the inherent ordinance of steering was lost with this implementation.

Empirical observations revealed that taking these rounding measures in the `.forward()` step introduced noisy gradient signals, impeding model learning. As a result, our adopted approach involves applying a sigmoid activation function to each action logit, with appropriate scaling for steering, allowing the model to train on continuous output. We also decided to overwrite the `__call__` method of our PlayerNet to do the appropriate rounding steps for both steering and braking, so as to not do these steps during training (by explicitly calling `.forward()` for each epoch) and only when our agent is on the playing field.

2.4 Imitation Learning Phase

Our imitation learning phase adopts a systematic approach. Utilizing a drill-based methodology, we curated a dataset comprising 200 matches (100 per team) characterized by randomized puck locations and velocities in each best-of-one encounter. With an average match duration of approximately 65 frames (terminating upon puck contact), our dataset encompasses approximately 13,000 frames of gameplay.

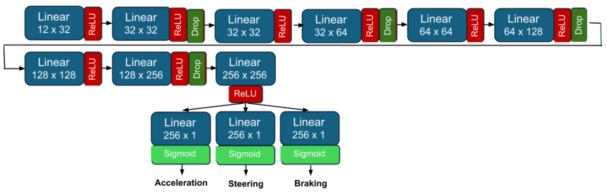

Figure 3: PlayerNet model neural network architecture consisting of 4 blocks each containing linear layers and ReLU activation functions

Our team tested a multitude of ways of sending gradient update signals to our model, paying particular attention to our loss calculation. After testing various output configurations, we adopted the smooth L1 loss function, implemented as `torch.nn.SmoothL1Loss`, for computing the loss across all actions within a batch in a single step. This decision was motivated by the favorable properties of SmoothL1Loss, which align with the requirements of our training task. Unlike mean squared error (MSE), SmoothL1Loss demonstrates robustness to outliers by treating large errors more akin to L1 loss through absolute error computation, while resembling MSE for smaller errors. This nuanced training environment facilitated the generation of more effective gradient signals compared to a reliance on MSE alone.

2.5 DAgger Training

Following the imitation learning phase, our state agent transitions to DAgger training, where it engages in 2v2 matches against other agents to refine its performance. Frame data is collected during these matches aganist varius AI opponents provided by the TAs, capturing the state of our agent and the soccer match at each timestep. For each frame, we take our state agents' current state as well as the state of the soccer match to calculate `jurgen_agent`'s opinion on what it should have done instead.

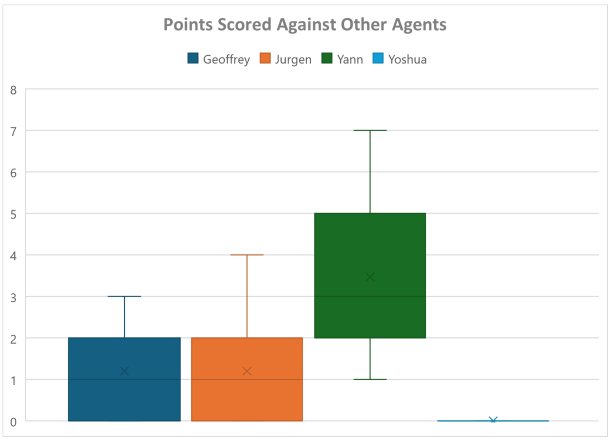

Figure 4: Scored goal distribution against other TA agents

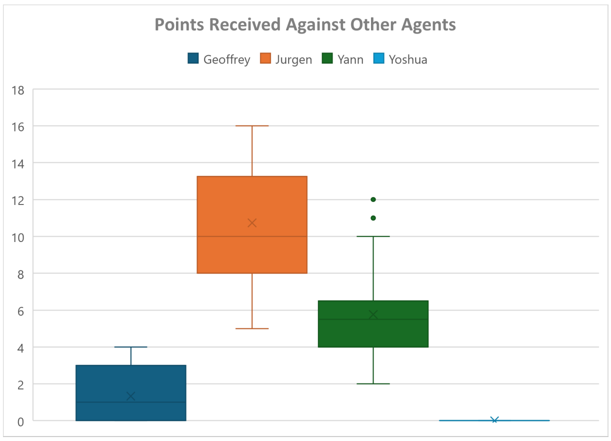

Figure 5: Goal score distribution received against other TA agents

However, this training phase presents challenges as our state agent exhibits erratic behavior, deviating from its learned driving drills. One prevalent issue is the tendency for the agent to collide with the playing field walls, resulting in exaggerated steering commands from `jurgen_agent`. Consequently, our state agents tend to over-learn from these scenarios, exhibiting persistent behaviors such as continuous left or right steering, regardless of contextual features. This phenomenon often manifests as "donut" behaviors, where the kart navigates in circular paths in pursuit of the puck.

Another approach was to truncate the frames in our matches, similar to how we did our drill-based dataset. This however also proved problematic as we chose not the randomize the puck location or velocity for these matches to get them as close to the real matches as possible. This produced "always drive straight" or "dummy" behaviors. We tried augmenting the hyper parameters of our training loop in many different ways, but the problem of the state agent either drifting donuts or flooring it straight was a persistent issue with our approach for which we could not overcome.

3 Results

Our objective for the fully-trained agent was to achieve a performance benchmark of scoring twenty or more total goals against opponent agents within a single test cycle. Each test cycle encompasses eight games played against each of the `jurgen`, `yann`, `yoshua`, and `geoffery` agents, resulting in a total of thirty-two games.

During evaluation, our player agent demonstrated considerable variability in its performance with respect to this objective. Scores ranged from as low as zero goals in thirty-two games to achieving up to seven goals in the same number of games.

Against `geoffrey_agent` and `jurgen_agent`, our player agent scored an average of 1.2 goals per test cycle. Against `yann_agent`, our agent scored an average of 3.5 goals per test cycle. The `yoshua_agent` pursues opponent players rather than the puck, and thus no goals were scored against this agent due to its constant interference with the player agent.

The subsequent section delves into alternative or supplementary methodologies that could augment the foundation established by the imitation learning and DAgger frameworks in this project.

4 Alternative Methods

4.1 Reinforcement Learning

We abstained from integrating a reinforcement learning phase into our training regimen due to the unstructured, trial-and-error nature inherent in true reinforcement learning (Ryan Sullivan, J K Terry, Benjamin Black, John P. Dickerson, 2022).

Had we pursued this training trajectory, it would have been necessary to devise a reward space for our player agent model. This reward space would consist of all game states, actions, and outcomes for which our player agent would be either positively rewarded (for desirable outcomes) or negatively rewarded (for undesirable outcomes). Within the reinforcement learning framework, the player agent's objective is to maximize its cumulative reward by actively seeking positive rewards while mitigating negative ones.

Crafting a comprehensive reward space demands meticulous attention, as unexpected learning patterns may emerge when the player agent discovers behavior policies that yield satisfactory cumulative rewards but deviate from the intended incentivized behavior envisioned by the designer.

4.2 Alternative Oracles

Our training protocol exclusively used `jurgen_agent` as our expert oracle. We could have experimented with other agents such as `yann`, `yoshua`, or `geoffery`.

Each of these agents employs distinct features extracted from the game state, and utilizes a unique deep neural network architecture to determine its output vector of actions per frame. Consequently, employing any of these alternative agents for either imitation training or DAgger training would lead to a player agent exhibiting different behavioral patterns compared to one trained exclusively with `jurgen` as its expert oracle.

Alternatively, a composite oracle could be devised by amalgamating outputs from several of the aforementioned agents. In this scenario, the output vector of the composite oracle would be a weighted average of the outputs from its constituent agents. However, this approach may encounter challenges during training, as the player agent could receive conflicting information when presented with a game state frame. The diverse output vectors of the constituent agents might attenuate one another when averaged, resulting in ambiguous guidance for the player agent on how to act.

4.3 Feature Selection and Filtering

Our feature selection process encompasses all features utilized in the training of `jurgen_agent`, augmented by an additional feature crucial for our "hit the puck" drill during the imitation step. However, further optimization could enhance our feature set's effectiveness in minimizing loss. Two prominent methodologies for feature selection, forward selection, and backward elimination, warrant consideration. Forward selection initiates with no features and progressively incorporates the most significant feature at each iteration, whereas backward elimination commences with all features and iteratively removes the least significant feature.

Alternatively, employing principal component analysis (PCA) presents another avenue for enhancing our feature set. PCA facilitates the reduction of the feature set into principal components ordered by their variance explanation within the dataset. This process effectively filters out components with low variance, which may not contribute significantly to our training task.

4.4 Input Normalization

The features in our dataset vary significantly in scale. For example, the position of the kart and puck spans from -60 to 60, whereas the angle of the puck relative to the goal line ranges from -1 to 1. To address these discrepancies, we could have normalized our features by subtracting the mean and dividing by the standard deviation of each feature. This standardization process will compress the range of input values, potentially enhancing the model's performance during training by providing a more uniform scale.

References

Zhao Chen and Darvin Yi. 2017. The game imitation: Deep supervised convolutional networks for quick video game AI. *CoRR*, abs/1702.05663.

Philipp Krähenbühl. 2024. Final project - SuperTuxKart ice hockey. Accessed on 4/1/2024.

Varun Ramesh and Eduardo Torres Montaño. 2017. Neuralkart : A real-time mario kart 64.

Ryan Sullivan, J K Terry, Benjamin Black, John P. Dickerson. 2022. Cliff Diving: Exploring Reward Surfaces in Reinforcement Learning Environments. Accessed on 4/21/2024.