

# Deep Learning Spring 2024: Final Project

Ananya Bhattacharyya, Chetan Kumar Verma,  
Satish Kumar Singh, Shiva Shankara Hirisave Srinivasa

University of Texas, Austin

## Introduction and Motivation

The aim was to design an automated agent capable of engaging in competitive gameplay against a pre-trained expert in a 2v2 match within the SuperTuxKart Ice Hockey environment. The agent can learn from the data provided by the environment, such as the location of the puck, opponents, goals, et cetera, either through processing images or states of players, with the objective being to score more goals than the opponents by maneuvering the puck into the opponent's goal.

While the approach of an image processing agent has access to a full 3D view of the world (Figure 1), it suffers from a need for higher computational resources, locating a tiny puck over large distances, having blind spots over certain pixels of the image, and the inability to use interesting strategies (such as reversing). Also, any disparity between the projected aimpoint and the puck's location on the screen introduces an error in learning.



Figure 1: Two images from left are from image processing agent. The agent finds it hard to spot the puck in the middle image. The third image is from a state-based agent. [1]

The approach of training the agent by learning from all the players' states was taken up since it was an interesting problem to address, given the large state and action space with sparse rewards of SuperTuxKart Ice Hockey to guide the agent towards the goal. Numerous approaches have been explored for tackling tasks like this one, often proving to be quite challenging.

SuperTuxKart Ice Hockey exhibits a Markov Decision Process (MDP) [2], and each player interacts with the environment, making sequential actions to maximize rewards. States represent the various situations an agent can find itself in,

while actions are the choices available to the agent in each state. Transition probabilities convey the likelihood of moving from one state to another after taking a particular action. Rewards serve as immediate feedback to the agent, indicating the desirability of its actions and guiding it toward achieving its goals. The policy acts as a strategy for the agent, determining which actions to take in different states and influencing its decision-making process.

The agent's goal within an MDP (Figure 2) is to uncover the best policy that maximizes its expected cumulative reward over time, enabling it to learn and make effective decisions in uncertain environments. The MDP exhibited by SuperTuxKart Ice Hockey leads to Reinforcement Learning (RL) as an obvious choice.

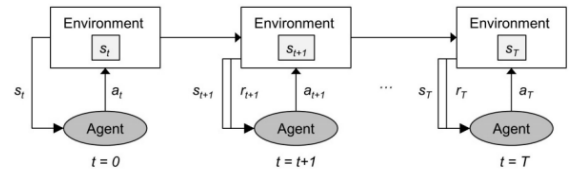


Figure 2: The agent-environment interaction in a Markov decision process (MDP) [3]

State-based learning is a control problem as opposed to a prediction problem and involves navigation, path planning, decision-making in uncertain scenarios, and learning long-term strategies. Reinforcement learning (RL) algorithms are versatile and can be readily applied to this problem where traditional approaches struggle. It explores various strategies and refines them based on feedback from the environment. Its notable successes in playing Atari games [4], [5], Chess [6][7], Go [8], and Dota 2 [9] demonstrate the versatility and effectiveness of RL in mastering complex gaming environments.

While traditional RL algorithms excel in many scenarios, they often need help with environments featuring large or continuous state and action spaces. Deep Reinforcement Learning (DRL) combines the decision-making principles of reinforcement learning with the function approximation capabilities of deep learning. In DRL, a deep neural network typically represents the policy or value function, enabling the agent to navigate complex environments with high-

dimensional states or actions. This fusion of deep learning and RL has catalyzed advancements in motion imitation, which is relevant to the current problem.

DQN (Deep Q Networks) was explored and presented with good possibilities, which significantly improves the Q-learning algorithm for learning optimal policies. Q-learning aims to learn the Q-value function,  $Q(s, a)$ , which is the expected cumulative reward of reacting with  $a_t$  given state  $s_{t-1}$  and following the optimal policy thereafter. The most straightforward improvement that Deep Neural Networks (DNNs) can offer over Q-learning is to approximate the Q-value function  $Q(s, a)$  using a neural network rather than storing a table of Q-values. This approach is known as Deep Q-Networks (DQN) [10]. DQN can handle large state spaces more efficiently and generalize across similar states, where tabular Q-learning is impractical due to the sheer size of the state space.

However, DQN relies on the temporal difference (TD) error signal, derived from the difference between predicted and observed rewards, to update Q-values. SuperTuxKart Ice hockey environment with sparse rewards, the TD error signal may not provide sufficient guidance for learning effective policies. This approach is also extremely sensitive to hyperparameters and suffers from stability and optimization issues, rendering it harder to train [11].


Proximal Policy Optimization (PPO), where a policy learned instead of a value function was explored. This has demonstrated success in mastering complex games, learning autonomous driving planners, controlling drones, et cetera, PPO exhibits a remarkable capacity to extract meaningful insights from a modest number of environmental samples, fostering efficiency in learning. One of its distinctive traits lies in its resilience to hyperparameter variations, mitigating the need for exhaustive fine-tuning efforts. Moreover, PPO's innate parallelizability streamlines the training process, harnessing computational resources effectively. And was therefore chosen as the first algorithm to be implemented.

## Methods

There are several challenges for training an agent to play Ice Hockey using RL:

1. Unlike traditional machine learning methods, the input data is not assumed to be independent and identically distributed. Predicting the action based on the kart's current position/state affects subsequent states and actions.

$$\text{i.i.d.: } p(\mathcal{D}) = \prod_i p(y_i|x_i)p(x_i)$$


  
 output  $y_1$  does not change  $x_2$

Equation 1: i.i.d property of ML input data [12]

2. The successful outcome for the agent is whether a goal has been scored. This feedback is sparse in every other timestep, which makes it hard for the agent to learn optimal paths without ground truths in each timestep.
3. While only the outcome of the task is defined, the agent needs to explore a huge number of states and actions to maximize the reward.

Before exploring the implementation of reinforcement learning algorithms for SuperTuxKart Ice Hockey, learning parameters and the environment commonly used need to be defined.

### Determining the state and action space

While the SuperTuxKart Ice Hockey setting provides the agent with the location of all the karts in a 2D space (the third dimension is ignored), the velocity of the kart, the location of the goal et cetera, it's transformed into a critical set of features (Table 1) which captures the complete state of the game, aligns to the completion of the task more closely (scoring goals) and also following the MDP property. To avoid exploding or vanishing gradients all the features are normalized.

The range of values these features can take was determined by running the SuperTuxKart game and capturing the maximum values seen over multiple episodes. *It was apparent that the agent needed to learn over a huge state space.* A glimpse of a few features:

State	State space (floating point values)
Agent's kart center in x coordinate	[-47.0, 47.0]
Agent's kart center in y coordinate	[-70.0, 70.0]
Agent's kart direction in radians	[-3.14, 3.14]
Agent's kart direction to puck in radians	[-3.14, 3.14]
Difference in angle between kart's direction and kart's direction to puck	[-1.0, 1.0]
Puck's position in x coordinate	[-47.0, 47.0]
Puck's position in y coordinate	[-70.0, 70.0]
Puck's distance from goal in x coordinate	[-47.0, 47.0]
Puck's distance from goal in y coordinate	[-70.0, 70.0]

Table 1: State space of the agent for Ice Hockey Game.

The agents react to control the kart to the above states seen in the environment in every timestep to accomplish the overarching objective. A few of the actions provided by SuperTuxKart: Nitro and Drift, are ignored initially to simplify learning. The brake is included since it allows for the kart to

reverse, which can be used to move the puck in the opposite direction when the agent is in an unfavorable position. This results in a mixed-action space (Table 2), which complicates the learning process.

Action	Action space
Steering, used to change direction. -1.0 is to the far left and 1.0 to the far right.	$[-1.0, 1.0]$
Acceleration, used to change the speed of the kart. 0.0 is a complete stop, 1.0 is the maximum speed.	$[0.0, 1.0]$
Brake, used to reverse the kart. Applied only when acceleration is 0.0	True/False

Table 2: Action space of agent for Ice Hockey Game

### Simulation for Training: Gym Environment

Reinforcement learning algorithms learn from a defined reward provided by the environment, which is not available in SuperTuxKart Ice Hockey. Therefore, a simulated training environment is created using the Gym framework [13]. The RL algorithm interacts with the Gym environment by providing actions and receiving states and rewards. The Gym environment internally encloses the SuperTuxKart Ice Hockey simulation and defines a reward function.

The Gym environment plays the actions of the algorithm as one of the players and an AI or human as the opponent. It can simulate a 1v1 or a 2v2 game with randomized ball locations and randomized initial states. Gym can also simulate parallel environments playing multiple matches simultaneously. The agent in each timestep calls *step()* of Gym and calls *reset()* to start a new episode (Figure 3).

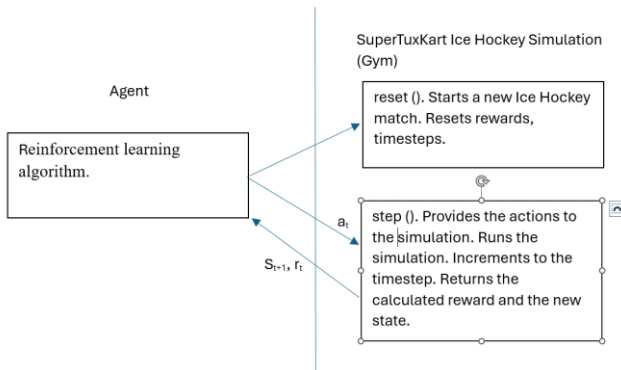


Figure 3: Gym environment setup returning state and reward each timestep and also provides a simulation. [14]

### Learning the Policy

Since policy gradient methods directly derive the policy  $\pi_\theta$  as opposed to estimating the value function first, exploration was done in search of effective on-policy algorithms. Games in similar problem space to Ice hockey were surveyed, yielding RoboCup[15] Soccer agents, and

RocketLeague[16], which have more complex state and action space. These use Proximal Policy Optimization (PPO) (Figure 4) and have state-of-the-art implementations in competitive tournaments.

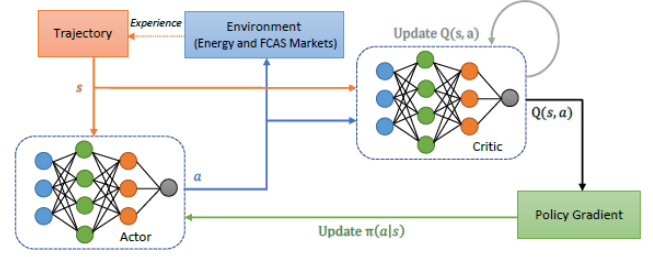


Figure 4: Proximal Policy Optimization [17]

PPO tends to need fewer samples from the environment, learns in a stable pattern, is easy to implement, is less sensitive to hyperparameter choices, is easily parallelizable, and importantly, handles continuous and discrete action spaces as well as large state and action spaces.

However, reward shaping proved to be too difficult in the Ice Hockey setup. An attempt was made to have the agent's kart without any opponent go near a stationary puck, using the equation:

$$\begin{aligned} \text{Reward}_t &= -\log(\text{Euclidean distance between kart and puck}) \\ \text{Reward}_t &= \text{Reward}_t + (\text{Reward}_t - \text{Reward}_{t-1}) \end{aligned}$$

The idea was to provide a higher reward as the kart got closer to the puck using the logarithm function and adding cumulative reward momentum. A logarithm function also reduces the dimensionality.

After training 50,000 timesteps, the kart was found only circling the puck [18] but never going close to it. Foreseeing the exponential difficulty in shaping the reward function to score goals in the presence of opponents, it was decided to imitate an existing expert policy and then learn the reward function through inverse reinforcement learning.

### Imitating a policy

A policy  $\pi_\theta$  is learned through imitation from expert demonstrations of policy  $\pi$  using supervised learning. The following assumptions can be made and relaxed from an RL algorithm setup:

1. The input data is treated as independent and identically distributed.
2. Each data point receives a ground truth (state-action pair from  $\pi$ ) to calibrate the policy  $\pi_\theta$ .
3. The training objective is to reduce the loss for each timestep rather than accomplish a task.

$$\max_{\theta} \sum_{(a_t, s_t) \in D} \log \hat{\pi}_{\theta}(a_t, s_t).$$

Equation 2: Loss function to train the policy. [19]

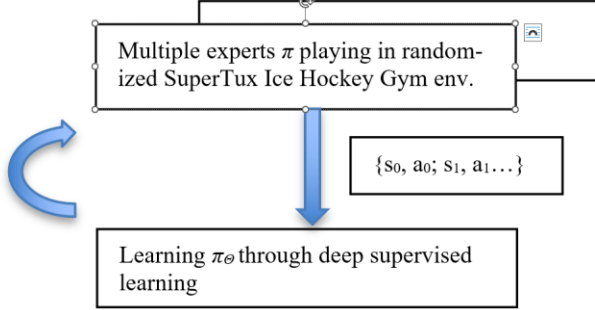


Figure 5: Architecture implemented for imitation learning. [20]

Multiple SuperTux Ice Hockey Gym environments are initialized, each with a randomized initial state and puck location. The environment is configured to play the expert against an AI. The environment is reset once a goal is scored, or the episode completes. The expert generates trajectories with state-action pairs. The trajectories from multiple environments are batched together and fed to a feed-forward neural network (Figure 5).

The loss on the neural network is calculated to maximize the similarity between the probability action output distribution of policies  $\pi$  and  $\pi_{\theta}$ .

$$p_{\text{data}}(\mathbf{o}_t) \approx p_{\theta}(\mathbf{o}_t)$$

Equation 3: Mismatch between the action's probability distribution of the expert and the policy being trained. [21]

This implementation excelled to a certain extent, but the i.i.d assumption poses a challenge in the MDP process, causing a distributional shift (Figure 6), where actions influence future actions. Minor errors are compounded quadratically, leading the agent to uncharted states and the model to be unable to correctly predict and course correct.

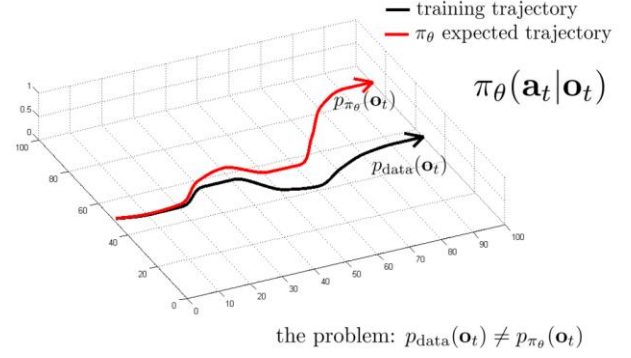


Figure 6: Minor errors in prediction lead to major trajectory differences since the expert can't predict the policy environment states. [21]

Given the huge state and action space in SuperTux Ice Hockey, the model didn't perform well when trained with 50,000 steps. [22].

The problem can be mitigated by training with a lot more data, which can lead to the policy  $\pi_{\theta}$  to converge. However, this is not optimal, given that the algorithm has live access to the policy  $\pi$  to correct actions.

### Dataset Aggregation

To mitigate the covariate shift in action distribution, the policy  $\pi_{\theta}$  is fed data from its environment augmented with expert demonstrations. This aims to improve the policy by improving the input data as opposed to making changes to the policy itself. The policy is trained with data it encounters during inference and expert data with decaying  $\beta$ .

#### Algorithm 1: Imitation with Data Aggregation

$D = \{s_0, a_0; s_1, a_1 \dots\}$

$\beta = 1$

**for**  $i = 1$  **to**  $N$  **do**

    Generate  $D_{\theta}$  querying policy  $\pi_{\theta}$ .

    Generate  $D$  querying expert policy  $\pi$ .

    Replace  $D$  with  $D_{\theta}$  with based on  $\beta$ .  $\beta = 1$  doesn't replace any action.  $\beta = 0$  replaces all actions

    Train  $\pi_{\theta}$  using  $D$ .

    Linearly decay  $\beta$ .

**return**  $\pi_{\theta}$ .

This iterative algorithm improves (Figure 7) robustness to new states and environments, reduces bias by using data from the environment where it learns data from the expert, and encourages exploration of new states, improving generalization. The learning model increasingly weans off policy  $\pi$ .

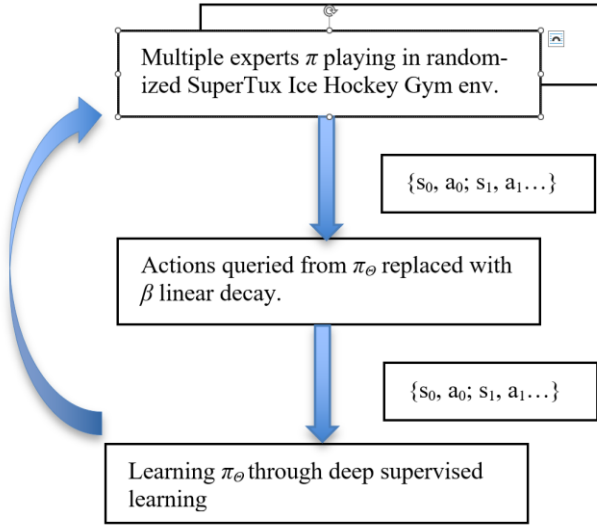


Figure 7: Architecture implemented for imitation learning using data aggregation. [23]

Much success was achieved with Data aggregation. The agents could be trained successfully to score goals against opponents, details are discussed under ‘Results’.

Data aggregation does a better job since it trains the policy with the data that the player sees during inference. Data aggregation alleviates the need for extremely accurate models and large data collections.

### Learning the reward function

The agent encounters an exhaustive list of states, and it quickly becomes tedious to design and train a reward function. They would also need to understand the weights for each state and action. Inverse reinforcement learning learns a reward function and a policy. However, this is something that will be taken up in subsequent trials.

### Metrics

**Loss Function:** Entropy was used as the loss function. For a random variable  $X$  with  $K$  discrete states, Entropy  $H(X)$  is defined as follows:

$$H(X) = \sum_{k=1}^K (-1) * p(k) * \log(p(k))$$

Equation 4: Entropy formula

Where  $p(k)$  is the probability of occurring  $k$ th event of the probability distribution. In other words, entropy is the negative of the sum of the probability of each event multiplied by the log of the probability of each event.

**Reward Function:** For episode “e” the reward function is defined as follows:

$$R(e) = \text{Goals scored by agent} - \text{Goals scored by opponent}$$

The reward function directly correlates with the ultimate objective of the task. Consequently, we monitored the mean and standard deviation of the reward function throughout the training process.

## Results

### Training

The agent is trained [reference to methods section] within an online Gym environment, leveraging the available state agent experts (including Geoffrey, Jurgen, Yann, and Yoshua) [24]. This approach allowed the agent to learn across a diverse range of scenarios, ensuring that its training was not limited to a specific set of sampled data. The agent is trained against an AI opponent in a 1vs1 match, and two models are trained (one for the blue and one for the red team) for policy network.

Each training batch consisted of 512 samples. The maximum timesteps per episode was 3000, choice made to enable the agent to explore more during each episode and encounter more random states. To enhance the stability of the training process, the agent was concurrently trained using five Gym environments. Models were trained on a single machine equipped with an Nvidia GeForce GTX TITAN X GPU.

For optimization, Adam optimizer was used with the default set of hyperparameters. L2 normalization was applied to regularize the weights of the neural network.

### Improving Training: Single vs Multiple Experts

Initially, the agent was trained with all available state agent experts, including Geoffrey, Jurgen, Yann, and Yoshua. However, during this phase, agent struggled to track the puck in a straight line (Figure 8). Subsequent analysis of the trajectories of both the experts and agent revealed that agent failed to replicate the behavior of multiple experts.

Consequently, we proceeded with the remainder of the training process utilizing the Jurgen expert exclusively. Jurgen exhibited superior performance among all state agents, as evidenced by the results of multiple tournaments on local grader evaluations.



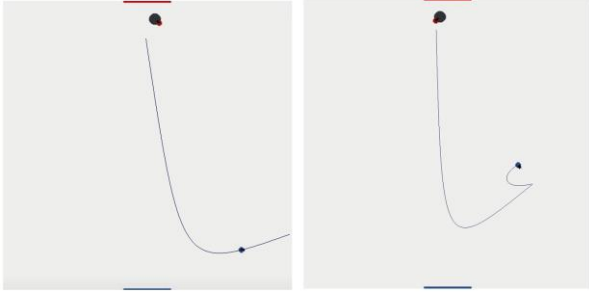


Figure 8: **(Left)** picture shows the trajectory (training with 100k steps) followed by agent (Blue) with continuous action space [25], **(Right)** picture shows the trajectory (training with 100k steps) followed by agent when trained using multiple experts [26].

### Improving Data: Continuous vs discrete action space

The acceleration data for the Jurgen agent over 100,000 steps was logged. Figures (9, 10) illustrate the distribution of this data across varying numbers of bins. As the number of bins increases, the data within each bin decreases significantly, resulting in highly imbalanced training data. This analysis underscores the challenge of learning within the agent's continuous action space. Therefore, we discretized the acceleration space to ensure that most of the information within this distribution is captured, providing sufficient data for meaningful agent training.

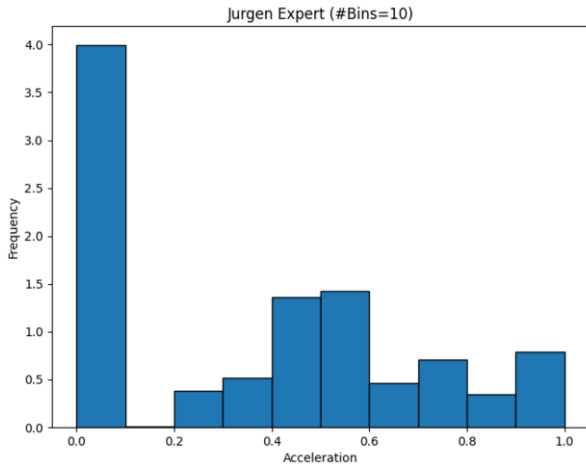


Figure 9: The histogram (Number of **bins=10**) represents the distribution of **acceleration** space for **Jurgen** expert over 100k steps

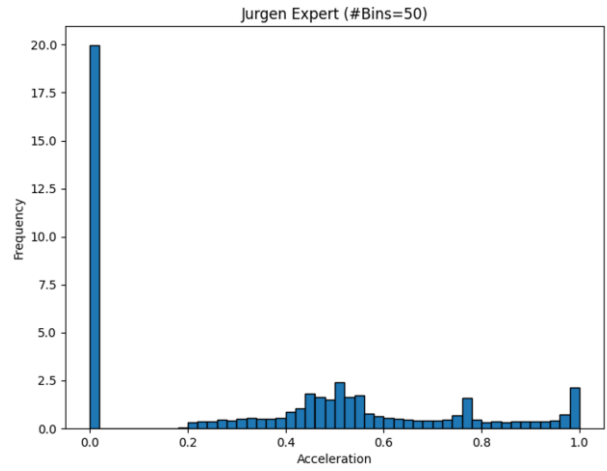


Figure 10: The histogram (Number of **bins=50**) represents the distribution of **acceleration** for **Jurgen** expert over 100k steps

The following analysis corresponds to the observed poor performance of agent, as evidenced by the absence of goals scored (goals scored=0), characterized by low values of acceleration and steering angle when operating within the continuous action space (Figure 8, add video reference). Consequently, we trained agent using a discrete action space, following this approach:

1. Discretizing the "acceleration" space into 90 bins (Experimenting with varying the number of bins = {10, 30, 50, 70, 90}, it was found that using 90 bins resulted in a better-performing agent).
2. The Jurgen expert outputs a discretized value for the "steering angle," eliminating the need to discretize the steering space.
3. The "brake" space was already discretized.

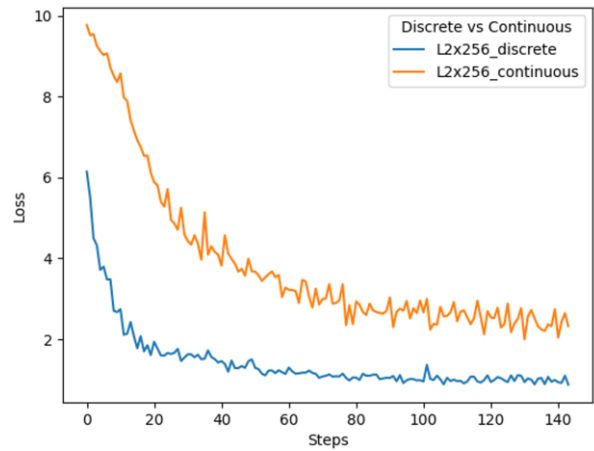


Figure 11: Plot compares the loss on using continuous and discrete action space (for base configuration, **L2x256**: 2

layers, 256 hidden units per layer and tanh activation function) during the training process.

### Improving Deep RL: Tuning Hyperparameters

To assess the significance of various components within the model, the base configuration was systematically varied. This configuration comprised a fully connected neural network with 3 layers, each consisting of 256 hidden units, utilizing the tanh activation function after every linear layer. And employing an AI opponent and the Jurgen expert. The impact of these variations on key metrics, including the impact on loss (Figure 12), mean reward (Figure 13), and the standard deviation of reward (Appendix) was measured.

Training was conducted for 500,000 steps or 6 hours for each configuration, with each training step requiring approximately 0.04 seconds to complete. The findings of this experiment are as follows:

- 1) Number of hidden units (per layer): Increasing the number of hidden units to 512 (refer to loss and reward curves for the following configurations: L2x256\_tanh and L2x512\_tanh) improved the learning capacity of the model, resulting in lower loss and higher mean reward throughout the training.
- 2) Number of layers: The size of the network can also be increased by augmenting the number of layers to 4 (refer to loss and reward curves for the following configurations: L2x512\_ReLU and L4x512\_ReLU), which again resulted in lower loss and higher mean reward throughout the training.
- 3) Activation function: Using the ReLU activation function (refer to loss and reward curves for the following configurations: L2x512\_tanh and L2x512\_ReLU) improved the mean reward for the training of the neural network along with a slight improvement in training loss.
- 4) Batch Normalization: Introducing batch normalization (refer to loss and reward curves for the following configurations: L2x512\_ReLU and L2x512\_ReLU\_BN) before each activation function resulted in an improved mean reward of the network.

Overfitting occurred upon further increasing the model's capacity. The best set of hyperparameters for the policy network (4 layers, 512 hidden units per layer, ReLU activation function, and batch normalization) improved the loss and mean reward throughout the training process.

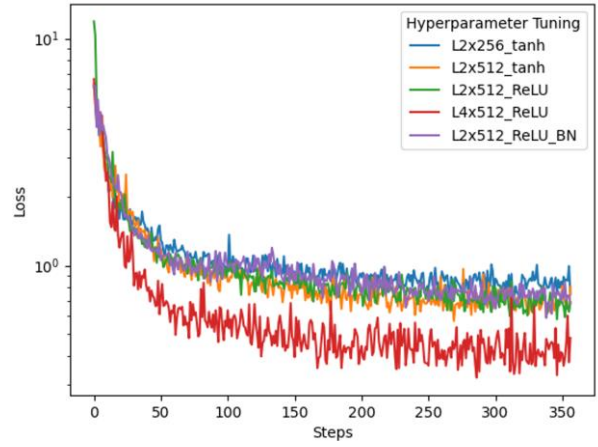


Figure 12: Plot (log scale) compares the **loss** for different set of hyperparameters against the base configuration (**L2x256\_tanh**: 2 layers, 256 hidden units per layer and tanh activation function) when agent is trained against **AI opponent**.

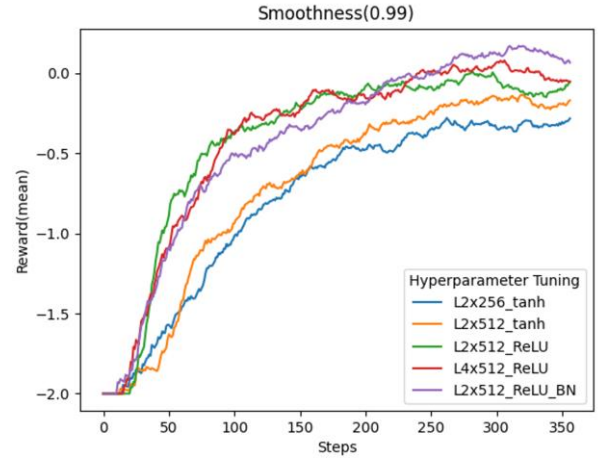


Figure 13: Plot (with smoothness=0.99) compares the **mean reward** for different set of hyperparameters against the base configuration (**L2x256\_tanh**: 2 layers, 256 hidden units per layer and tanh activation function).

Furthermore, transitioning from the AI opponent to the Jurgen opponent resulted in improvements across all metrics (refer to Appendix for more details), underscoring enhanced training efficacy and subsequent generalization.

### Local grader results

The agent was evaluated on a local grader, which essentially tests the agent's ability to play against all available state agent experts in a tournament, each consisting of 8 matches. The agent was evaluated 5 times on the local grader to measure the stability of the trained agent on a given machine. The numbers in Table 3 depict the results

of these tournaments against all the experts, highlighting the robustness of trained agent on a given machine. We argue that the variance in results across the three machines is due to underlying differences in hardware architecture and low-level kernel implementations.

Runs	GPU (Linux)	CPU (Windows)	CPU (M3 Mac)
1	66	72	85
2	69	72	85
3	69	72	85
4	69	72	85
5	69	72	85

Table 3: The data shows the result (score out of 100) of local grader (across 5 runs) for agent (base configuration).

### Online grader results

The agent with base configuration (Tested the base configuration to detect any issue in pipeline and ended up with the same configuration for this task) was evaluated on an online grader, which essentially tests the agent's ability (similar to the local grader evaluation) to compete against state-of-the-art experts in a tournament, each consisting of 4 matches. In total, the agent scored 18 goals (Table 4) against the expert, which translates to a score of 88/100.

	Expert	Goals scored	Score board
1	Jurgen	2	0:2 0:0 1:2 1:0
2	Yann	4	0:2 2:1 1:0 1:2
3	Geoffrey	5	2:0 0:0 1:0 2:0
4	Yoshua	7	1:0 3:0 1:1 2:0

Table 4: The above data shows the goals scored (and score board) by agent (base configuration) against online experts across the four games. The number in the table translates to **88 (out of 100)** score on online grader.

## Conclusion

In this work, a state-based agent was engineered using existing methods from the literature, capable of competitive gameplay against state-of-the-art pretrained experts. While it was successful in this task, given the trained agent's performance in the online grader (18 goals scored across 16 matches), more experiments are needed with Inverse reinforcement learning.

Optimism surrounds the future of RL-based methods/algorithms for solving complex problems that are not feasible using traditional machine learning algorithms. The next incremental improvement would involve enhancing the solution for this task by replacing the AI opponent with the Jurgen opponent (Appendix), replacing the categorical distribution with the Gumbel-SoftMax distribution, and

substituting the current approach with a reward-based reinforcement learning method.

## References

- [1] SuperTuxKart 0.9.3. URL: <https://www.youtube.com/watch?v=zMSo7blGMGg&t=288s>
- [2] Richard S. Sutton and Andrew G. Barto: Reinforcement Learning: An Introduction. URL <https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRL-Book2ndEd.pdf>
- [3] Duarte, Fernando & Lau, Nuno & Pereira, Artur & Reis, Luís. (2020). A Survey of Planning and Learning in Games. Applied Sciences. 10. 4529. 10.3390/app10134529. URL [https://www.researchgate.net/publication/342579276\\_A\\_Survey\\_of\\_Planning\\_and\\_Learning\\_in\\_Games](https://www.researchgate.net/publication/342579276_A_Survey_of_Planning_and_Learning_in_Games)
- [4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmille. Playing Atari with Deep Reinforcement Learning. URL <https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf>
- [5] Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, D.; and Blundell, C. 2020a. Agent57: Outperforming the atari human benchmark. *arXiv preprint arXiv:2003.13350*, 2020.
- [6] Deep Blue (chess computer): [https://en.wikipedia.org/wiki/Deep\\_Blue\\_\(chess\\_computer\)](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer))
- [7] D.Silver, T.Hubert, J.Schrittwieser, I.Antonoglou, M.Lai, A.Guez, M.Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- [8] David Silver, Demis Hassabis. AlphaGo Blog: URL <https://deepmind.google/discover/blog/alphago-zero-starting-from-scratch/>
- [9] Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; Jozefowicz, R.; Gray, S.; Olsson, C.; Pachocki, J. W.; Petrov, M.; de Oliveira Pinto, H. P.; Raiman, J.; Salimans, T.; Schlatter, J.; Schneider, J.; Sidor, S.; Sutskever, I.; Tang, J.; Wolski, F.; and Zhang, S. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- [10] Robert Kozma, Cesare Alippi, Francesco Carlo Morabito. Artificial Intelligence in the Age of Neural Networks and Brain Computing (Second Edition), 2024. URL: <https://www.sciencedirect.com/topics/computer-science/deep-q-network>
- [11] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep Reinforcement Learning That Matters. Proceedings of the AAAI Conference on Artificial



Intelligence, 32(1). URL <https://ojs.aaai.org/index.php/AAAI/article/view/11694>

[12] CS 182: Lecture 14: Part1: Imitation Learning. URL [https://youtu.be/kGc8jOy5\\_zY?t=118](https://youtu.be/kGc8jOy5_zY?t=118)

[13] Gym Documentation. URL <https://www.gymnasium.dev/index.html>

[14] Gym: Make your own custom environment. URL [https://www.gymnasium.dev/content/environment\\_creation/](https://www.gymnasium.dev/content/environment_creation/)

[15] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pp. 340–347, New York, NY, USA, 1997. Association for Computing Machinery. ISBN 0897918770. doi: 10.1145/267658.267738. URL <https://doi.org/10.1145/267658.267738>.

[16] On the Verge of Solving Rocket League using Deep Reinforcement Learning and Sim-to-sim Transfer. URL <https://arxiv.org/abs/2205.05061>

[17] M. Anwar, C. Wang, F. de Nijs and H. Wang, "Proximal Policy Optimization Based Reinforcement Learning for Joint Bidding in Energy and Frequency Regulation Markets," 2022 IEEE Power & Energy Society General Meeting (PESGM), Denver, CO, USA, 2022, pp. 1-5, doi: 10.1109/PESGM48719.2022.9917082. URL <https://ieeexplore.ieee.org/document/9917082/references#references>

[18] Agent trained using PPO algorithm for 50,000 timesteps. URL <https://www.youtube.com/watch?v=MdOevC5bXYc&list=PLlp7QYKYXwnPE7WmomT5ODQFBDKenHr72&index=8>

[19] CS 182: Imitation learning. Page 5. URL <https://cs182sp21.github.io/static/discussions/dis8.pdf>

[20] Imitation learning. Section 10.2. URL [https://web.stanford.edu/class/cs237b/pdfs/lecture/lecture\\_10111213.pdf](https://web.stanford.edu/class/cs237b/pdfs/lecture/lecture_10111213.pdf)

[21] Sergey Levine. CS 294-112: Deep Reinforcement Learning. Behavior Cloning. URL [https://rail.eecs.berkeley.edu/deeprl-course-fa17/f17docs/lecture\\_2\\_behavior\\_cloning.pdf](https://rail.eecs.berkeley.edu/deeprl-course-fa17/f17docs/lecture_2_behavior_cloning.pdf)

[22] Agent trained using behavior cloning algorithm for 50,000 timesteps. URL <https://www.youtube.com/watch?v=e1xe74eRhM4&list=PLlp7QYKYXwnPE7WmomT5ODQFBDKenHr72&index=8&pp=iAQB>

[23] Imitation learning. Section 10.3. URL [https://web.stanford.edu/class/cs237b/pdfs/lecture/lecture\\_10111213.pdf](https://web.stanford.edu/class/cs237b/pdfs/lecture/lecture_10111213.pdf)

[24] Final project - SuperTuxKart ice hockey. URL [https://www.philkr.net/dl\\_class/homework/final/](https://www.philkr.net/dl_class/homework/final/)

[25] Trajectory followed by agent on training with continuous action space. URL

<https://www.youtube.com/watch?v=FYH0hiq20a4&list=PLlp7QYKYXwnPE7WmomT5ODQFBDKenHr72&index=1>

[26] Trajectory followed by agent on training with multiple experts. URL [https://www.youtube.com/watch?v=Ycl\\_cDJPdXs&list=PLlp7QYKYXwnPE7WmomT5ODQFBDKenHr72&index=2](https://www.youtube.com/watch?v=Ycl_cDJPdXs&list=PLlp7QYKYXwnPE7WmomT5ODQFBDKenHr72&index=2)

## Appendix

### Hyperparameter Tuning

The [figure 14](#), shows the standard deviation of reward for different configurations that were used to find the best set of hyperparameter for the agent's policy network. All of the configurations show similar values at the end of training.

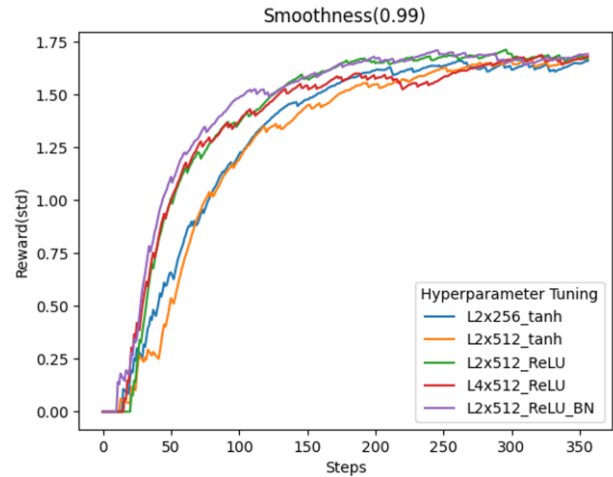


Figure 14: Plot (with smoothness=0.99) shows the **standard deviation of reward** for the different configurations when agent is trained against the AI opponent.

### AI and Jurgun opponents

The following figures ([15](#), [16](#), [17](#)) shows the improvement over all the metrics (loss, mean reward, and standard deviation of reward) when agent is trained against the Jurgun opponent instead of AI opponent.

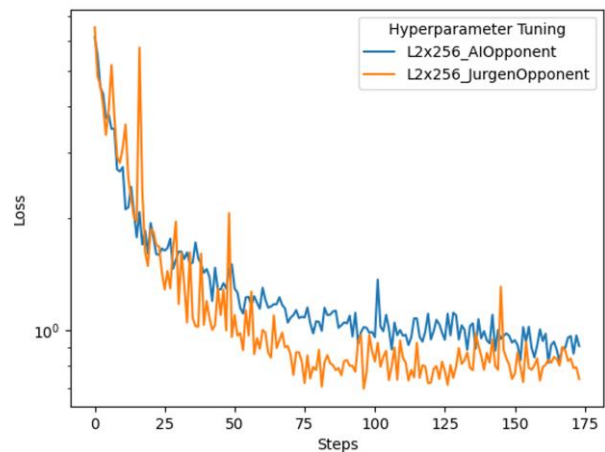


Figure 15: Plot (log scale) shows the improvement in **loss** when agent is trained against **Jurgen opponent** (instead of AI opponent) for base configuration (**L2x256**: 2 layers, 256 hidden units per layer and tanh activation function).

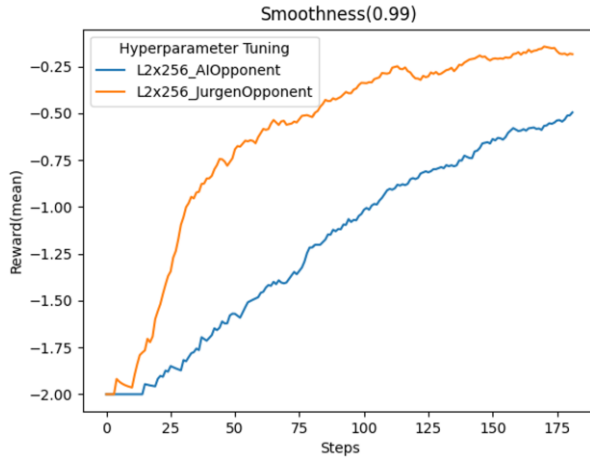


Figure 16: Plot (with smoothness=0.99) shows **large mean reward** for base configuration (**L2x256**: 2 layers, 256 hidden units per layer and tanh activation function) when trained using **Jurgen opponent** as opposed to AI opponent.

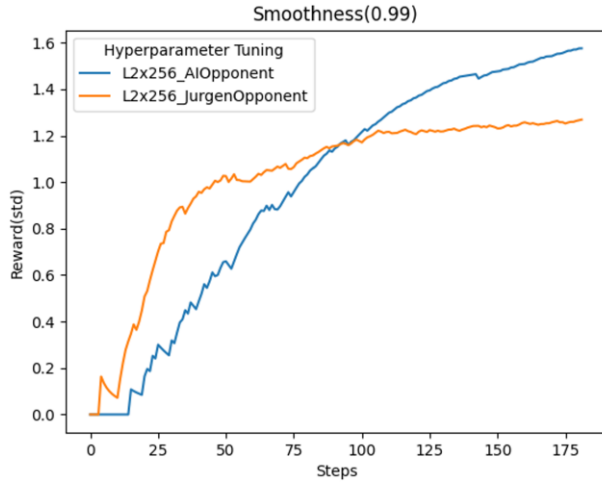


Figure 17: Plot (with smoothness=0.99) shows **lower standard deviation of reward** for base configuration (**L2x256**: 2 layers, 256 hidden units per layer and tanh activation function) when agent is trained against the **Jurgen opponent** as opposed to AI opponent.