AI 394D DEEP LEARNING 2024: REINFORCE MAVERICKS

FINAL PROJECT TECHNICAL REPORT

· **Shubham Gupta**[*]
shubhamgupta@utexas.edu

· **Jean Del Rosario Peguero**
jcd4284@utexas.edu

· **Emmanuel Rajapandian**
emmanuel.rajapandian@utexas.edu

ABSTRACT

We design an automated agent to play SuperTuxKart Ice Hockey which is a game featuring a vast state space, a diverse action space, and sparse rewards, presenting a highly formidable challenge. The objective of the agent is to maximize goal scoring in any difficulty and achieve victory in the match if possible. Our approach involves using imitation learning, combining both Behavioural Cloning and DAgger, to mimic other agents and learn the optimal strategy for playing the game. We employ REINFORCE on top of our best imitation agent to adjust the variables of the agent's policy in an approach that increases the likelihood of actions that result in higher rewards i.e., an effective goal-scoring strategy. Our system is designed to exploit potential simplifications in this complex environment, with the ultimate aim of creating proficient players. We train a neural net model, inspired by the principles of imitation learning, to support a controller network to play ice hockey. Our system is state-based, focusing on the state of the game rather than the visual input from the player's field of view.

1 Introduction and Motivation

In this ever-growing field of artificial intelligence and games, the task of creating a proficient agent capable of mastering complex game environments remains an enduring challenge. Our project dives deep into the intersection of these two domains by focusing on the old-school SuperTuxKart Ice Hockey game. The motivation behind our project flows from an ambitious goal to tackle multiple challenges simultaneously. We aim to design an automated agent that not only excels at goal scoring in SuperTuxKart Ice Hockey but also demonstrates adaptability across various difficulty levels of AI opponents and achieves a positive outcome in each game. This objective reflects our commitment to push the boundaries of state-based agents in deep learning and overcome the challenges of the game environment's sparse rewards and complicated state space.

To address these challenges, we explored and focused on a two-fold strategy. Initially, we used imitation learning, where an agent replicates the expert's strategy using a classifier after receiving training data on the states and actions of the demonstration Faraz et al. [2019]. Through imitation, the agent was trained to mimic the behavior of the Jurgen agent providing a solid baseline for subsequent learning. The REINFORCE method was implemented over our imitated agent to increase the anticipated cumulative rewards i.e., scoring more goals in the Ice Hockey game.

Our project standing at the intersection of deep learning research and gaming, offers significant insights into the sequential learning approaches in tackling complex game environments such as SuperTuxKart. By prioritizing the state of the game instead of the visual image input, our system is capable of identifying simplifications and carving the way for the creation of proficient state-based player agents. Through imitation learning and REINFORCE, our project aims to create a simplified and holistic approach with the ultimate goal of creating adaptive agents capable of mastering the SuperTuxKart Ice Hockey game challenges.

[*]Ordered alphabetically by last name — University of Texas, Austin; Deep Learning; Spring 2024.

2 Methods

In our exploratory phase of the project, each of us explored both state and image based options. However, owing to time and resource constraints, we finalized on state based agent approach. As part of the deep dive, we compared the test agent(s) provided by the Instructor, Geofrey, Jurgen, Yann, and Yoshua to a local grader. We iterated the grader 4 times over each agent. Then, we calculated the agent's goals, wins, losses, and draws. Finally, we selected the agent that scored the most goals (since wins do not add up to more goals scored).

Agent	Jurgen	Yann	Yoshua	Geoffrey	Score (100)
Jurgen	6	11	14	15	94
Yann	1	8	7	5	66
Yoshua	1	7	0	1	29
Geoffrey	1	7	7	6	66

Table 1: Average goals scored by each agent (row) against the Local Grader

The Jurgen agent was by far the best-performing agent out of the four. Given the Jurgen agent's capability to score more goals than any other agent, we decided to make it our "expert" for the model to emulate and gain understanding.

2.1 Training Dataset

We create training datasets by utilising our agent's unique extract features function in combination with its behaviour to extract features from the states of the player, opponent, and ball. A vector value comprised of the agent's action's braking, steering angle, and acceleration is the result for every observation. For this purpose, we created a function that orchestrates the collection of data by setting up and executing multiple tournament simulations involving different combinations of agents. We generated the data using the following steps:

- Execute tournament simulations using the `tournament.runner module`.
- Run commands for various agent match-ups and execute them in parallel.
- Collect data from these simulations which is then used for training and evaluating the performance of agents.

The approach to generating training datasets was to simulate game data by executing tournament simulations of the Jurgen agent pitted against other agents in a competitive setting and a specified number of games, which is called Behavioural Cloning [Stéphane et al., 2011]. To enhance this, we also applied DAgger, by running our imitation agent against other agents and then collecting which action would the expert agent, Jurgen, would have taken. By simulating multiple matches between various combinations of agents, we gather data on agent performance, identify strengths and weaknesses, and then subsequently improve our agent strategy through training. To speed up the process of running multiple simulations, the code utilizes parallel execution by employing a process pool with multiple processes to execute the tournament simulations concurrently. This parallelization helped us in speeding up the data collection process. During each simulation, the game states and actions are recorded, providing training data for the imitation learning agent.

Once the game simulations are completed, the recorded game states are processed and formatted for training the imitation learning agent. Recorded game states are loaded and formatted for training by extracting relevant features and labels. Expert labels for the training data are obtained by predicting actions using the pre-trained expert agent `(jurgen_agent)`. After loading the game states, the data is shuffled to introduce randomness and prevent bias during training. This step ensures that the imitation learning agent learns from a diverse set of examples, enhancing its adaptability and generalizing its learning. Subsequently, the data is prepared for training after converting it into tensors.

For the REINFORCE-based approach, we collected training data in a 2 vs 2 game setting, focusing on updating the policy network of the first car agent exclusively. This decision originated from the challenge of integrating information from both agents into a single neural network for simultaneous updating, forcing us to opt for a simpler strategy. We engaged the agent in a series of game episodes using our parallelized approach, where we played 10 games, with 5 games played against the Jurgen agent and another 5 against the Yann agent. All games were conducted with the agent acting as the second player.

The data collection and processing stages are the most crucial parts of this project as they make or break the imitation agent to play in this complex game environment. Our effective way of data collection and organization lays a proper foundation for training a robust agent capable of mimicking the expert Jurgen.

2.2 Model Architecture

Our deep learning model's general architecture is largely inspired by the Multi-Layer Perceptron (MLP), a feed-forward network comprising an input layer, multiple hidden layers, and an output layer. MLPs are characterized by their ability to learn complex nonlinear relationships in data, owing to the presence of multiple layers and activation functions. In our case, the ReLU activation functions between the layers introduce nonlinearities, enabling the model to understand complex patterns in input imitation data. Known as "Universal Approximators" [Hornik et al., 1989], MLPs with just a unit hidden layer, substantial number of neurons within that layer have the capacity to represent any finite input-output mapping problem.

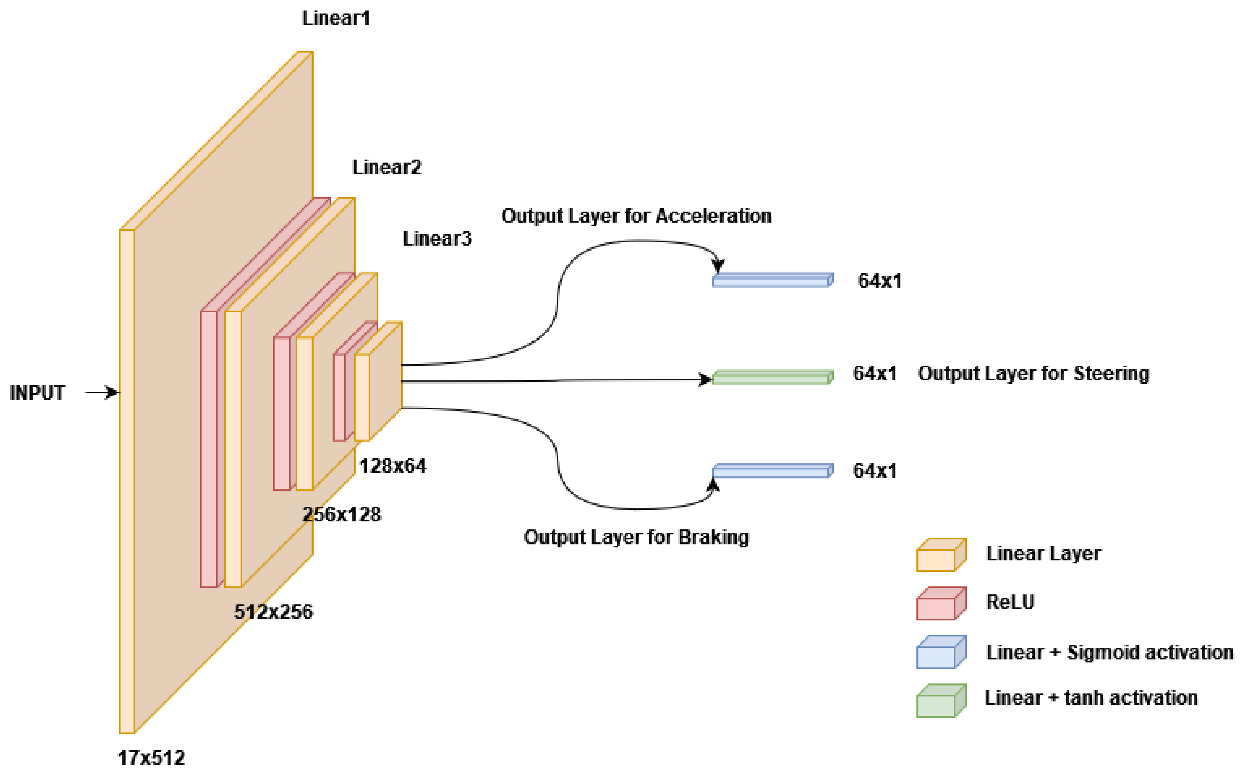


Figure 1: Model Architecture

The structure consists of several layers that are fully interconnected, with a Rectified Linear Unit (ReLU) activation function deployed between the layers to introduce non-linear characteristics. The neural network contains four hidden layers **[512, 256, 128, 64]**, followed by dropout regularization. The dropout regularization helps prevent over-fitting by arbitrarily setting a fraction of input units to zero during training.

The network has three separate output layers: `acceleration`, `steering`, and `brake`, each responsible for a specific action parameter in the environment. The acceleration and brake outputs are constrained to the scope [0, 1] and thus employ the sigmoid activation function to compress the output to this scope, representing probabilities of acceleration and braking. The steering output is already constrained to the scope [-1, 1] due to the use of the tanh activation function, which maps the output to this scope.

2.3 Game Environment Class

We create a class that encapsulates the functionality and behavior of the game environment in which the reinforcement learning agent interacts. It serves as a connector between the RL algorithm and the SuperTuxKart game, providing methods to reset the environment, take steps, calculate rewards, and extract relevant information from the game state. The class sets up various other parameters such as team names, the number of players, the maximum score, and other settings related to the game environment. We chose the `icy_soccer_field` as our track for the environment to learn and play. Overall, the Game Environment class acts as an interface between the RL algorithm and the game environment, providing the necessary functionality for training and evaluation of the RL agent within SuperTuxKart Ice Hockey.

2.3.1 Functions within Environment

We define a `reset` function within this class which is responsible for resetting the game environment to its initial state. This involves tasks such as initializing the game race, setting player configurations, and updating the game state. Additionally, it handles cleanup from previous episodes, such as stopping the race and clearing the environment.

The `step` function takes care of the interaction between the RL agent and the game environment. It takes actions provided by the agent, advances the game state, and returns observations, rewards, and information about whether the episode has ended. Within this method, actions from both teams are processed, the game state is updated, and rewards are calculated based on the outcome of actions and the current game state.[Toro Icarte et al., 2022]

The `reward_fn` defines how to calculate the reward for a given state transition. This function considers various factors such as the game score, the ball's location, and the proximity to opponent AI's goal.

It rewards our agent when it successfully scores a goal. This encourages the agent to actively pursue scoring opportunities. The reward for scoring a goal is **+100**, indicating a successful outcome.

$$R_{\text{goal}} = \begin{cases} \text{goal\_reward (100)}, & \text{if agent scores goal successfully} \\ 0, & \text{otherwise} \end{cases}$$

It also penalizes the agent when the opponent scores a goal. This discourages defensive lapses and motivates the agent to prevent the opponent from scoring. The penalty for conceding a goal is **-100**, indicating an unfavorable outcome.

$$R_{\text{penalty}} = \begin{cases} \text{goal\_penalty (-100)}, & \text{if agent concedes a goal during defense} \\ 0, & \text{otherwise} \end{cases}$$

We also factor the distance between the ball's location and the opponent's goal line. This encourages our agent to advance the ball in proximity to the opponent's goal line, raising our chance of scoring. The reward is higher when the ball is in close proximity to opponent's goal line and diminishes as distance increases.

$$R_{\text{proximity}} = \frac{|\text{max\_distance} - \text{distance\_to\_goal}|}{\text{max\_distance}}$$

We also consider the distance between the ball's location and our agent's own goal line. This discourages own goals and defensive vulnerabilities by penalizing the agent for allowing the ball to approach our own goal line.

$$R_{\text{proximity\_penalty}} = \frac{|\text{max\_distance} - \text{own\_distance\_to\_goal}|}{\text{max\_distance}}$$

$$\boxed{R_{\text{total}} = R_{\text{goal}} + R_{\text{penalty}} + 1.5 \times R_{\text{proximity}} - 1.5 \times R_{\text{proximity\_penalty}}} \quad (1)$$

Overall, this reward function provides a structured way to reward desirable behaviors and penalize undesirable ones, guiding our agent towards effective game-play strategies in the SuperTuxKart Ice Hockey environment.

2.4 Model Training

In our model training code, we have an Actor class which serves as the neural network model governing the policy in reinforcement learning. This model is tasked with mapping the state of the environment to actions such as acceleration, steering, and braking. The main training loop maintains the iterative update of the policy network using accumulated experience from interactions with the environment. Initialization sets up the training environment, including the policy network, and optimizer, alongside hyper-parameters such as the number of episodes, batch size, and learning rate. Episodes are generated by engaging with game environment based on our current policy, gathering sequences of the game states, game actions, and rewards from each episode.

The policy network is then updated based on the collected data to maximize expected returns. We repeat this process for a designated number of iterations. Our model, during training aims to maximize the expected returns, or cumulative discounted rewards, acquired from environment interactions. This is achieved by minimizing the negative log-likelihood of the actions undertaken by our policy, weighted by the returns associated with the actions.

Parameter	Value
Number of Episodes	100
Number of parallel Episodes	50
Optimizer(s)	Adam, AdamW, SGD
Model Iterations	50, 75, 100
Batch Size(s)	64, 128, 256
Learning rate(s)	3e−5, 1e−4, 1e−5
Ball locations	[0, 1], [0, -1], [1, 0], [-1, 0]

Table 2: Parameter(s) combinations used for training

Hardware(s) - Model Trained & Tested Upon
MacBook Pro M2, 8-core GPU, 10-core CPU
Windows 11, Nvidia RTX 1650Ti
Google Colab T4 GPU
Kaggle Nvidia Tesla P100 GPU
Runpod Ubuntu Linux, Nvidia RTX A4000
AWS EC2 g5.2xlarge Nvidia A10G GPU
MacBook M3 Pro, 14-core GPU, 11-core CPU

Table 3: Hardware Details

Overall, our model training process was a meticulous process aimed at updating the parameters of the model to build an effective policy for maximizing rewards within the environment of the SuperTuxKart Ice Hockey game.

3 Results & Performance

3.1 Local Grader Results

For our team, changing the data collection and exploring different algorithms such as imitation learning, Q-Learning, and REINFORCE [Bhatnagar, 2023] [Williams, 1992] on imitation learning offered various improvements in model performance. We optimised the agent for multiple in-game scenarios by fine-tuning each of them. We determined that a good model was one in which we outscored the other agents that were presented against our agent. In order for our algorithm to learn how to score goals from different puck start positions, it was essential to collect accurate data. Even though our training took hours by running the Ice Hockey Engine, and simulating hundreds of games, our effective data collection allowed the RL agent to learn and replicate the Jurgen agent's performance.

In our initial observation into the implementation of the REINFORCE algorithm, we noticed a trend wherein prolonged training adversely affected the model's efficiency. Initially, during early iterations of model's training loop, improvements were noted in the model's efficacy. However, as the training continued, typically after 20 iterations, a distinct decline in model performance was noticed. This phenomenon is indicative of a critical threshold beyond which extended training leads to a degradation rather than enhancement of the model's capabilities. Following are the expected log returns from the first 3 episodes of our training of the RL agent on our machines.

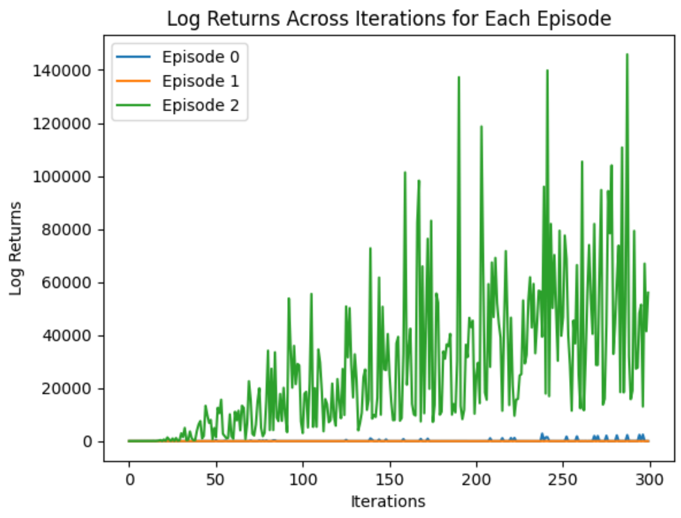


Figure 2: Expected Log returns across 300 iterations of each episode

For our best-performing agent locally, the best dataset we created was by playing against the Jurgen agent 5 times and against the Yann agent 5 times in which all the games were played with the agent acting as the second player. It was a 2 vs 2 game setting focused completely on the policy network update of the first car agent. Our next contribution to the

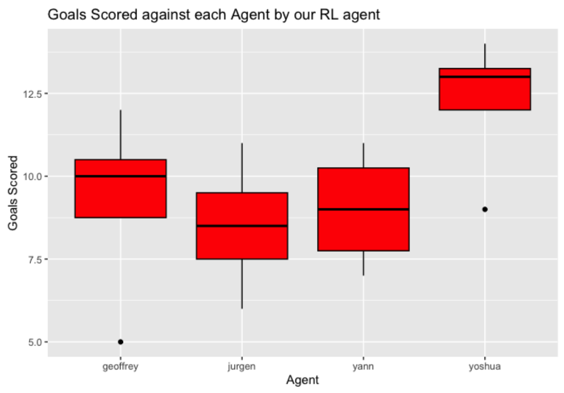
RL agent's success was our model architecture pooled with our reward function. Our first model was a simple MLP with just 2 hidden layers with dropout regularization, which resulted in rather poor performance against the local grader agents scoring a maximum of 60/100 after hours of training. Adding more hidden layers with a proper number of nodes, removing the dropout layer, and tweaking the optimizer increased the performance. So just by modifying the architecture and data collection our model showed considerable improvements. To create the required data, our model makes use of the same collection of 17 features that are present in the feature extraction function of the Jurgen agent.

Here is a table presenting the outcomes of our model's performance across different datasets, hyper-parameters, and methodologies.

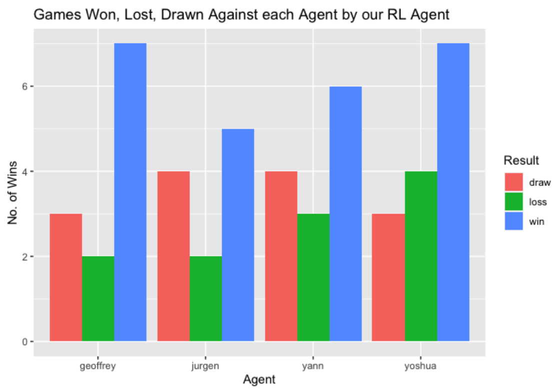
Model	Technique	Average Goals	Local Grader Scores
2-Layer Plain MLP	Imitation Learning w/ initial data collection	0.68	79, 60, 67, 65, 72
4-Layer MLP w/ dropout	Imitation Learning w/ Increased batch-size	0.81	82, 88, 70, 85, 80
4-Layer MLP w/o dropout	REINFORCE on best Imitation agent	1.11	91, 94, 91, 94, 97
4-Layer MLP w/o dropout	REINFORCE w/ new data collection technique	1.28	100, 97, 94, 100, 97

Table 4: Results of running various models trained on multiple datasets through the local grader

We immediately noticed that increasing the hidden layers from 2 to 4 significantly improved the number of goals scored by our agent in the Local grader. Additionally, we noticed that grader results were varying on different machines, so we settled for a model with robust performances across all machines (all the packages were of the same version across different systems).



(a) Win, Loss, Draw Statistics against Local Grader Agents



(b) Goals Scored by our final Agent against the Local Grader Agents

3.2 Canvas Grader Results

One of the main reasons why we built and tested our model on multiple hardware was to make sure our model was robust in performance irrespective of the PyTorch compatibility in various systems as described in hardware details table 3. Our best model locally scored consistently above 90 on multiple machines 4 and hence we decide to upload the same `jit` file to the canvas grader to assess our RL agent's performance against tougher agent opponents. We were happy to observe that our agent did not deviate much from the the local grader scores and scored 16 goals against 4 agents in 4 games each. Following is the canvas grader output:

Agent	Goals Scored	Results
Geoffrey	7 goals in 4 games	(3:0 1:0 2:0 1:0)
Jurgen	2 goals in 4 games	(0:3 0:3 0:1 2:1)
Yann	4 goals in 4 games	(0:0 0:1 2:1 2:0)
Yoshua	3 goals in 4 games	(0:1 2:0 0:0 1:1)

Table 5: Canvas grader result of our best agent; Total: 82/100

4 Conclusions

In our project, we achieved success in development of our agent crossing 80-point threshold on the online grader, exhibiting reproducibility of results. Our methodology was centered on gathering training data within a 2 vs 2 game context, with a specific emphasis on refining the policy network of the primary car agent. The finalized model architecture is a sequence of fully interconnected linear layers, with ReLU activation. Remarkably, our agent had a commendable performance, averaging 1.28 goals per game against the local grader.

An issue our team encountered during this project was the time-intensive nature of model training, since we did not have access to high-performance GPU resources. Moreover, the lack of precise information regarding online grader machine, required us to build a comprehensive model which scores well across multiple computing machines.

Moving forward, avenues for enhancing our agent's capabilities include exploring U-Net architectures, adding residual and skip connections. Additionally, fine-tuning both the training dataset and model parameters could help us augment the agent's goal-scoring proficiency to even greater scales.

4.1 Future Enhancement & Other Explorations

In this section, we discuss future enhancements and other explorations related to our work on SuperTuxKart Ice Hockey. While our current methodology has yielded promising results, there are several avenues for improvement and further investigation. As a future enhancement we could scrutinize aspects such as player interaction, kart behavior optimization, and incorporating RL feedback to build a robust agent which can maneuver the game environment effectively.

4.1.1 Training Data

In our training data, we acknowledge the potential for further refinement in our methodology by incorporating game-play experiences where the agent assumes the role of the first player as well. This aspect presents an avenue for enhancing the robustness and adaptability of our trained agent by capturing a broader spectrum of game-play scenarios and dynamics.

4.1.2 Q-Learning

We have also attempted to employ a Deep Q-learning (DQN) approach to train an agent to play a soccer game. Our agent interacts with the game environment acquiring capability to take optimal actions to score goals and win the game. In our implementation we established an Agent class responsible for engaging with the environment, executing actions, and retaining experiences within a replay buffer. This agent's deep neural network estimates the predicted future reward for each action, which is its core principle. During training, the agent explores the environment using an epsilon-greedy strategy, balancing between random explorations and exploiting its learned knowledge. A separate target network is also used to improve the stability of training. This approach utilizes experience replay to improve training efficiency and a separate target network to stabilize the learning process.

References

Torabi Faraz, Warnell Garrett, and Stone Peter. Generative adversarial imitation from observation. In *ICML Workshop on Imitation, Intent, and Interaction, PMLR 97, 2019*, pages 3–6. ICML, 2019. URL `https://arxiv.org/pdf/1807.06158.pdf`.

Ross Stéphane, J.Gordon Geoffrey, and Bagnell J.Andrew. A reduction of imitation learning and structured prediction to no-regret online learning. In *14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*, pages 2–7. Carnegie Mellon University, 2011. URL `https://arxiv.org/pdf/1011.0686.pdf`.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. In *Neural Networks, Vol. 2*, pages 356–399. University of California, San Diego, 1989. URL `https://www.sciencedirect.com/science/article/abs/pii/0893608089900208`.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. In *Journal of Artificial Intelligence Research*. Vector Institute, Toronto, ON, Canada, 2022. URL `https://arxiv.org/pdf/2010.03950.pdf`.

Shalabh Bhatnagar. The reinforce policy gradient algorithm revisited. Indian Institute of Science, 2023. URL `https://arxiv.org/pdf/2310.05000.pdf`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning, Volume 8*, pages 229–256. Northeastern University, Boston, 1992. URL `https://link.springer.com/article/10.1007/BF00992696`.