Space Rats Neural Network Lab Report

Jeeva Ramasamy
Rutgers University
jeeva.ramasamy@rutgers.edu
Course: 16:198:520
Instructor: Professor Cowan

Abstract

In this project, we design and evaluate a neural network capable of estimating the remaining steps required for Bot 2 from the Space Rats Project to capture a space rat based on its knowledge base. Bot 2's knowledge base is represented as two 30 x 30 grid maps: one representing the potential positions of the bot and the other reflecting Bot 2's utility estimates for navigating the grid. The grid's layout features blocked outer edges and the same interior structure used in the Fire Extinguisher and Space Rats Projects. For this project, the grid layout remains fixed while the starting positions for the bot and the space rat are varied in each simulation. The neural network is trained and tested using simulation data generated at a space rat detector sensitivity parameter (α) of 0.15, which introduces realistic uncertainty into the detection process. The network's performance is evaluated by its ability to predict the number of remaining steps until the bot successfully captures the space rat. Results indicate that the network achieves reasonable predictive accuracy, demonstrating its potential to enhance bot strategies by providing informed estimates of task completion.

1 Introduction

Building on the utility-based strategies developed for Bot 2 in the Space Rats Project, this work investigates whether a neural network can predict the bot's remaining steps to capture the space rat using its knowledge base. Bot 2's knowledge base consists of two 30 x 30 grids—one representing the potential positions of the bot and the other capturing utility estimates based on the bot's strategy. The ability to estimate the remaining steps has significant implications: it could inform real-time decision-making and provide insights into the efficiency of navigation strategies.

2 Methodology

This section outlines the processes used to create the simulated dataset and design the neural network to predict the remaining steps.

2.1 Data Representation

The data used in this project captures the environment and state of the simulation at any given time, allowing the model to predict the number of turns remaining for the bot to reach the space rat. The data is structured as follows:

- **Input Features**: Two **30 x 30** grids of type float, where all walls are represented by 0. Both grids are aligned such that their dimensions and corresponding cells map directly to each other in the simulation environment. The walls are pre-determined and remain static throughout the simulation.
    - **Bot Potential Location Grid**: Each cell contains either 0 (no chance of the bot being at this location) or 1 (possible location for the bot). At the start of phase 1, all open cells contain 1, implying that the bot can be in any location. During phase 2 of the simulation, only one 1 exists in the grid, representing the exact location of the bot.
    - **Utility Grid**: Represents the utility values for the bot's decision-making.
- **Output**: An integer representing the number of turns left until the bot reaches the space rat. This value is calculated retroactively at the end of each simulation to be used for training, making this a supervised learning problem.

2.2 Dataset Generation

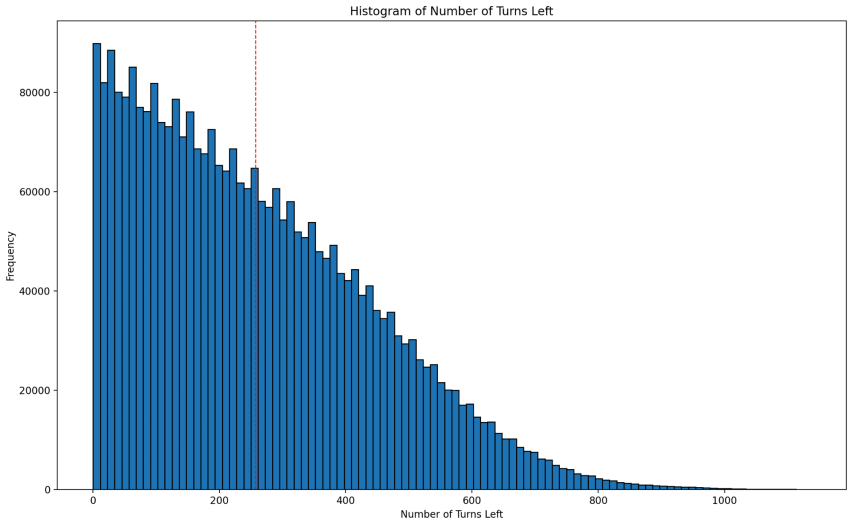
The space rat detector sensitivity parameter (α) was set to **0.15** due to the superior performance of Bot 2 at this value. Consistent grid generation at each run was achieved by setting the random number generator's seed to **520**, eliminating variations in starting positions and grid configurations.

For each simulated game, the bot's **knowledge base at every timestep** is included in the final dataset. This approach enhances the model's training by leveraging the complete set of runs, providing a more diverse representation of possible scenarios. The dataset is structured as a list of tuples in the following form:

> **[(bot_locations, knowledge_base, num_turns_left), ...]** where bot_locations and knowledge_base are 30 x 30 grids and num_turns_left is an integer

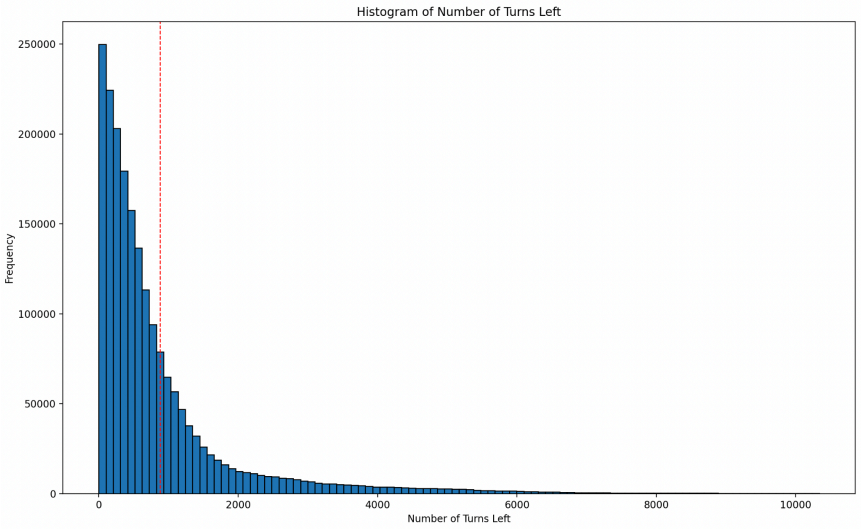
This dataset is split into **Training (80%)**, **Validation (10%)**, and **Testing (10%)** sets. This particular split is chosen because it ensures that the model has sufficient data for training while reserving a meaningful portion for validation and testing. The entire dataset is saved to a **compressed numpy archive** (.npz format), offering efficient storage and fast retrieval during training and evaluation.

- **Stationary Space Rat Case**: A total of **7,500** game simulations, or **3,021,059** entries, were generated for training and evaluation. The average number of turns left is approximately **257.29**.



Histogram of the Number of Turns Left for the Stationary Space Rat Case

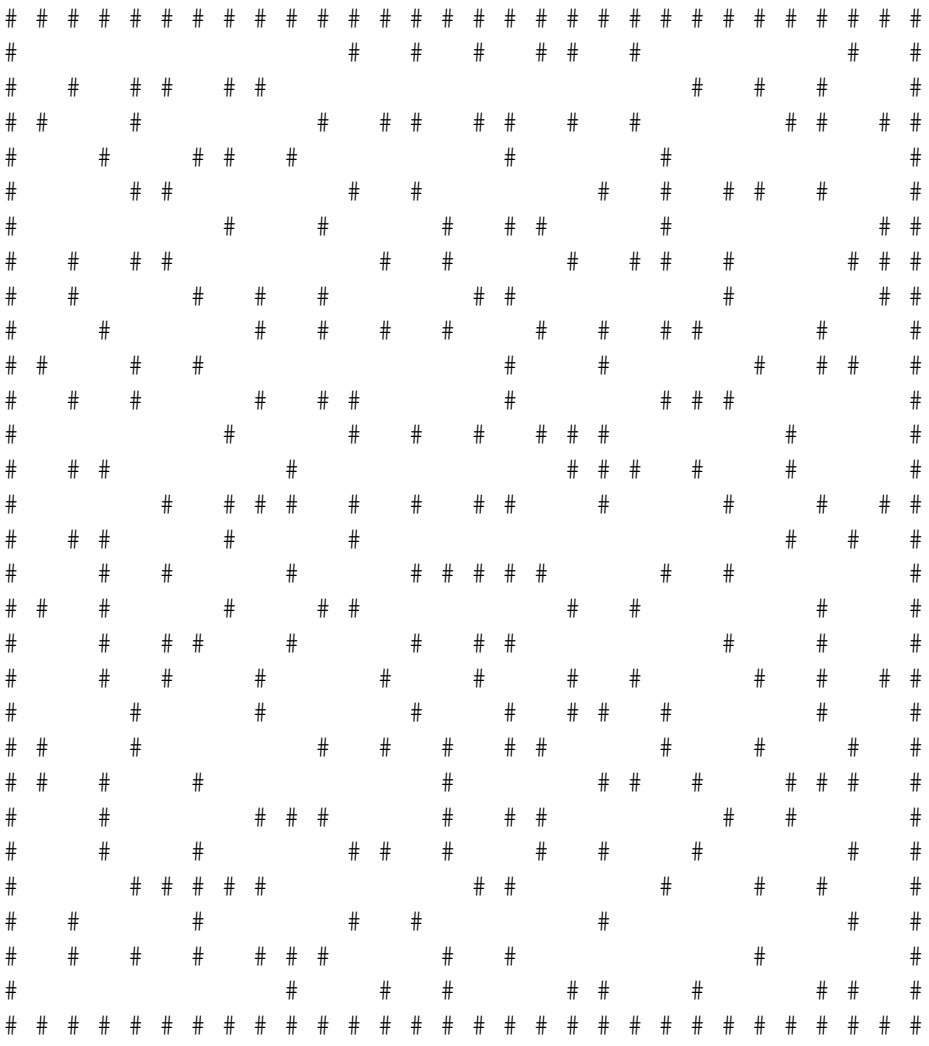
- **Mobile Space Rat Case**: A total of **2,500** game simulations, or **1,986,422** entries, were generated for training and evaluation. The average number of turns left is approximately **878.60**.



Histogram of the Number of Turns Left for the Mobile Space Rat Case

We can observe that the "Number of Turns Left" distribution is **right-skewed** for both cases, though it is much more severe in the Mobile Space Rat Case. These outliers are retained in the dataset because they represent valid and realistic scenarios within the simulation, capturing the complexity and variability of the problem domain. Removing these outliers could lead to a loss of critical information and potentially hinder the model's ability to generalize to challenging or extreme cases.

- **Grid Configuration with Obstacle and Path Layout (30 x 30)**



- **Computational Constraints**: After several attempts to increase the number of simulations, I ultimately settled on generating 7,500 entries for the stationary space rat case and 2,500 for the mobile space rat case. This decision was driven by limitations in CPU compute power and available RAM on my local machine. Using Google Colab was not a viable alternative, as it is optimized for GPU performance and offers less CPU power than the M1 chip. The discrepancy in the number of simulations arises because mobile space rat simulations are approximately three times longer than stationary ones on average. For more details on the computing environment, see Section 2.3.

2.3 Neural Network Design

The model is built to process spatial information from the input grids while effectively capturing patterns and relationships between the bot's potential locations and the utility grid. A combination of convolutional and fully connected layers is used to handle the high-dimensional input and produce a scalar output.

- **Model Architecture**: The neural network's architecture consists of convolutional layers and fully connected layers. The network begins with an **input batch normalization** layer followed by **two convolutional layers**, with **kernel sizes of 3x3** and filter sizes of **32** and **64** respectively, each paired with **batch normalization** and **ReLU activations**. These layers extract spatial features from the input grids. After the convolutional layers, the feature maps are flattened, and the data is passed through **two fully connected layers** with **64** and **32** units, respectively, each employing **ReLU** activation for non-linear transformations. A **dropout layer with a 50% rate** is included to prevent overfitting by introducing regularization. Finally, the network concludes with a fully connected layer that outputs a single scalar, representing the predicted number of remaining steps.
    - **Batch Normalization** is included to stabilize training by normalizing the activations across mini-batches, allowing the model to converge faster.
    - After extensive testing, using **larger filter sizes** for the convolutional layers (increasing from 32 to 64) and **reducing the number of neurons** in the fully connected layers (decreasing from 64 to 32) resulted in the lowest average error.
    - **Convolutional Layers**: Tested 1, 2, 3 → 2 was best (1 did not capture enough information, 3 started overfitting)
    - The numbers 32 and 64 (for both convolutional and fully connected layers) were chosen primarily due to **computational constraints**; adding more filters may improve performance.
    - **Kernel sizes of 3x3** are specifically chosen since the bot primarily receives information about its immediate surroundings using the space rat detector, which can be represented as a 3x3 grid. This choice is an attempt to model the bot's localized perception and decision-making process in the neural network.
    - **Fully Connected Layers**: Tested 1, 2, 3 → 2 was best (1 did not capture enough information, 3 started overfitting)
    - **ReLU** is utilized for its efficiency and performance; other activation functions, such as Tanh, did not lead to improvements.
    - The **dropout rate of 50%** is chosen to force the model to learn multiple high-level features rather than the specific noise in the training set. However, this high rate led to unintended consequences such as lower validation loss than training loss. See Sections 3.1 and 3.2 for more details.
- **Loss Function and Optimization**: The **Huber loss** function measures the difference between the predicted and actual number of turns left. This choice is due to the continuous nature of the output and the presence of a significant number of outliers. Using Mean Squared Error (MSE) in such cases would lead to biased predictions, skewed towards higher values of turns due to the influence of large outliers. The **AdamW optimizer** is employed for its adaptive learning rate capabilities that enable efficient convergence. AdamW is chosen over Adam since it offers better regularization by

decoupling weight decay from the gradient updates. This results in improved generalization and reduced overfitting.

- **Hyperparameter Tuning**: Key hyperparameters, such as learning rate, batch size, dropout rate, and the number of convolutional filters, were tuned through experimentation. Validation performance over several epochs serves as the primary performance indicator.
    - **Learning Rate**: Tested 1e-6, 1e-5, …, 0.01 → 0.001 was best
    - **Dropout Rate**: Tested 0.1, 0.2, …, 0.5, 0.6 → 0.5 was best
    - **Batch Size**: Tested 4, 8, 16, 32, 64, 128 → 64 was best
    - **Weight Decay (for AdamW)**: Tested 1e-6, 1e-5, 1e-4, 1e-3 → 1e-4 was best
    - **Number of Epochs**: Trained each model for 50 epochs to test performance → 20 was best (started overfitting after)
    - While further granularity could have been explored (e.g., testing learning rates of 5e-4 or batch sizes of 48 or 96), searching for minor improvements in validation error might not generalize well to the test set.
- **Training Procedure**: Training involves feeding the model batches of data and shuffling the order of inputs at each epoch to minimize the impact of data ordering. The best model is tracked and saved for testing.
- **Hardware and Computational Resources**: The training process was executed on a MacBook Air with an M1 chip and 16 GB RAM, which provided adequate support for CPU-based computations but limited GPU capabilities. Given these constraints, the dataset size was adjusted to ensure efficient memory usage, with simulations capped at 7,500 entries for the stationary case and 2,500 for the mobile case. The model is also designed to use lightweight architectures and batch processing to accommodate the hardware limitations without compromising predictive performance.

3 Results

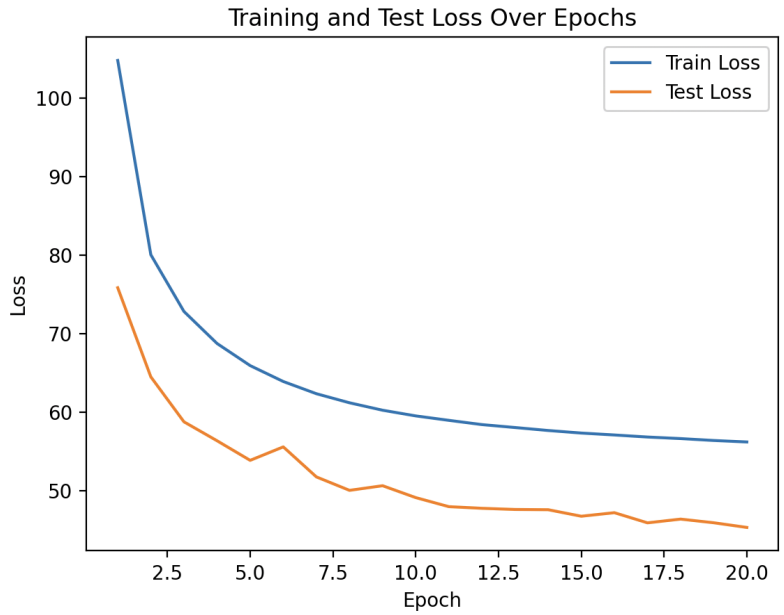
This section presents the outcomes of the experiments, analyzing the model's performance and comparing the results across different cases. Key metrics, visualizations, and observations are discussed to provide insight into the model's behavior and predictive capabilities.

3.1 Model Evaluation and Testing

The model with the lowest validation error during training is selected as the best-performing candidate for testing. This ensures that the chosen model generalizes well to unseen data while minimizing overfitting. The selected model is then evaluated on the **10% holdout test dataset**, which was not used during training or validation. Testing on this dataset provides an unbiased assessment of the model's performance in predicting the number of turns left in the simulation.

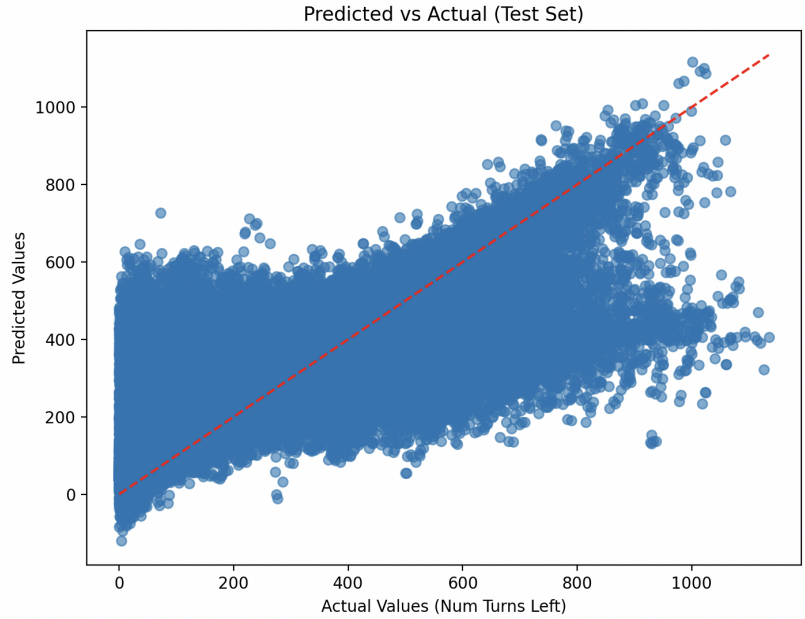
- **Stationary Space Rat Results**

  The stationary space rat scenario provides a controlled environment where the space rat remains fixed in position throughout each simulation. This simplifies the problem by removing dynamic elements, allowing the model to focus solely on predicting the bot's ability to navigate the grid and capture the rat.



Training and Test Loss Over Epochs for Stationary Space Rat Case

We can see in the graph above that the training and validation loss decreases over training time but starts to plateau after around **18** epochs. The gradual reduction in both losses suggests that the network is converging effectively without overfitting to the training set.



Scatter Plot of Predicted vs. Actual Number of Turns Left for Stationary Space Rat Case

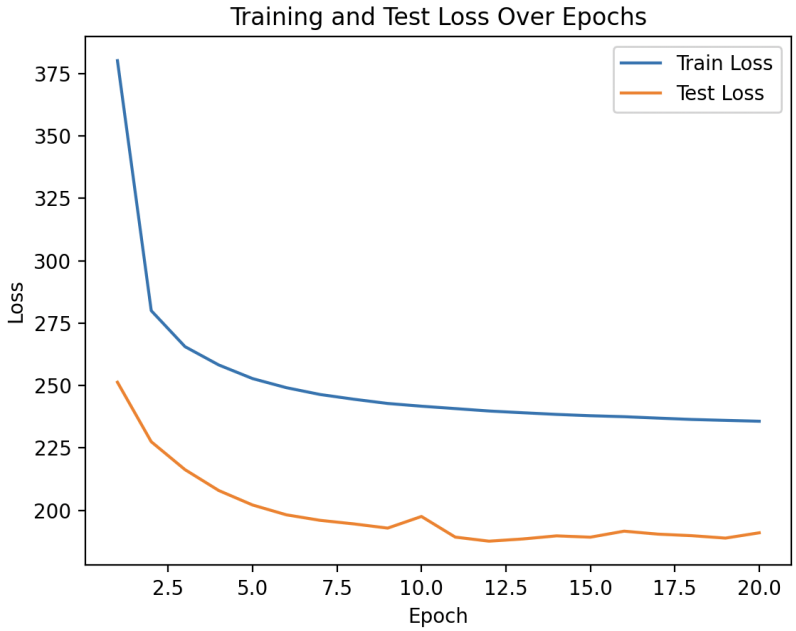
In the scatterplot above, the data points appear to fall into two distinct groups. The first group aligns closely with the diagonal, indicating that the neural network accurately predicts these cases with minimal error. This suggests that the network has successfully learned the underlying patterns or features that characterize this subset of the dataset.

The second group, however, demonstrates a significant deviation from the diagonal, with predictions clustering around the mean value of the dataset. This pattern suggests that the neural network is struggling to learn the properties of this group, possibly due to insufficient distinguishing features, limited representation of these cases in the training data, or a bias toward predicting values close to the mean.

The **final test loss on the holdout 10% dataset is 45.4746**, which is consistent with the validation test loss of the best model.

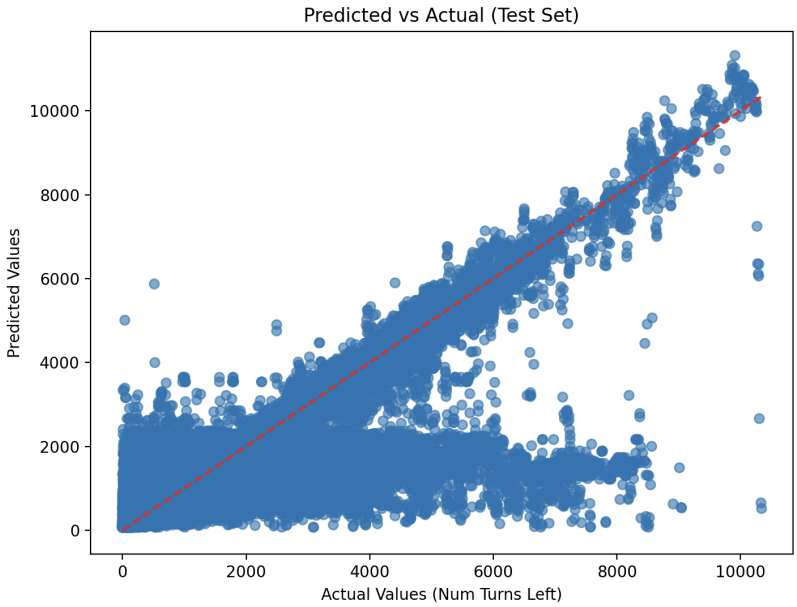
- **Mobile Space Rat Results**

  The mobile space rat scenario introduces a dynamic element to the simulations, as the space rat moves throughout the grid during each run. This adds complexity to the problem, requiring the model to account not only for the bot's navigation strategies but also for the unpredictable behavior of the moving target. This scenario better simulates real-world conditions, challenging the model to generalize its predictions to environments with continuous changes and increased uncertainty.



Training and Test Loss Over Epochs for Mobile Space Rat Case

We can see in the graph above that the training and validation loss decreases over training time but starts to plateau after around **12** epochs. The gradual reduction in both losses suggests that the network is converging effectively without overfitting to the training set.



Scatter Plot of Predicted vs. Actual Number of Turns Left for Mobile Space Rat Case

Like the Stationary Space Rat scenario, this scatter plot also shows two distinct groups. The first group is represented by points close to the diagonal, indicating accurate predictions where the model effectively captures the relationship between the inputs and outputs. The second group, however, deviates significantly from the diagonal, suggesting cases where the model struggles to generalize. In this group, predictions tend to cluster around the dataset's mean.

The **final test loss on the holdout 10% dataset is 189.6521**, which is consistent with the validation test loss of the best model.

3.2 Discrepancy Between Training and Validation Loss

In both the stationary and mobile space rat scenarios, the **validation loss is consistently lower than the training loss**. While this might initially appear counterintuitive, it is a logical outcome given the neural network's architecture. The final dropout layer, with a rate of 50%, forces the model to **randomly deactivate half of its neurons during training**. This constraint prevents the model from overly relying on specific neurons, encouraging it to learn more high-level features.

However, during validation and testing, the dropout layer is disabled, allowing the model to utilize its **full capacity**. This results in lower loss values on the validation and test sets, as the network operates without the constraints imposed during training. This design choice helps mitigate overfitting and enhances the model's ability to generalize to unseen data.

4 Discussion

The trained model has many implications and potential extensions, particularly its use as a decision-making tool for evaluating actions in the simulation environment. By leveraging the model's predictions, we can assess the quality of potential actions and analyze how closely the model aligns with the bot's actual behavior.

- **Using the Model to Evaluate Future Actions**: The trained model offers an opportunity to evaluate the expected outcomes of different actions the bot might take. For instance, predict the new state of the simulation if the bot moves up and evaluate the number of turns remaining according to the model. Similarly, we can predict and evaluate the new states if the bot moves down, left, right, or senses. By comparing the model's predictions across these potential actions, it is possible to determine which action minimizes the number of turns left, effectively identifying the optimal decision based on the model's learned behavior.
- **Agreement Between Model Predictions and Bot Actions**: We can also measure how often the model's suggested best action aligns with the bot's actual behavior in the simulation. High agreement suggests that the bot's decision-making aligns closely with the patterns learned by the model, indicating consistency and reliability in the simulation logic. Disagreements between the model and the bot can highlight situations where:
    - The bot's actions may be suboptimal.
    - The model may struggle to generalize in specific scenarios.
    - The model does not fully capture complex dynamics in the simulation environment.

By exploring the synergy between the trained model and bot actions, this method emphasizes the practical utility of machine learning in enhancing simulation-based decision-making systems. Future work could involve formalizing this evaluation framework and quantifying its impact on overall simulation performance. In addition, generalizing the model across different grid layouts would expand its applicability to a wider range of scenarios and challenges.

5 Conclusion

This project demonstrates the use of neural networks to predict the number of turns remaining in a maze-like simulation environment with 30x30 grid inputs. By leveraging a structured dataset of simulated cases, the model was trained to capture complex spatial and temporal relationships, achieving meaningful predictions for both stationary and mobile space rat scenarios. The analysis of results highlighted the strengths and limitations of the approach, including the challenges of generalizing to uncertain or highly dynamic environments. Furthermore, the potential application of the trained model as a decision-making tool showcases its versatility and opens pathways for future enhancements, such as real-time optimization of bot strategies.