Neural Analytics for Space Rat : Project 3 in Introduction to AI (CS-520)

Harsha Rajendra (hr458), Santhosh Janakiraman (sj1230)
Computer Science Department, Rutgers University
hr458@scarletmail.rutgers.edu, sj1230@scarletmail.rutgers.edu

1.0 Abstract

Building on the foundation of **Project 2**, In **Project 3** we developed a neural network model to predict the bot's actions required to locate and capture the space rat in a dynamic grid-based simulation of a ship. Using the probabilistic tracking framework from **Project 2**, the model predicts the number of moves and sensor actions needed to achieve the goal. It was designed and tested for accuracy and reliability, with performance evaluated through training and testing losses. Visualizations were used to analyze how the model's predictions align with or improve upon the bot's actions. This project demonstrates the potential of neural networks to enhance decision-making and strategies for tracking and capturing moving targets in complex environments.

2.0 Recap of Project 2

Our Project 2 involved designing a bot to navigate a 30 x 30 grid environment representing a ship and locate a target - the space rat. The ship had approximately 60% open cells and all outer edge cells blocked, required the bot to start in an unknown location due to memory loss caused by a cosmic blast. The task was divided into two phases: **localization** and **tracking and capturing the space rat**. In the localization phase, the bot used blocked cell sensing and movement to iteratively determine its original starting position. Once localized, the bot transitioned to the second phase, where it relied on a probabilistic detector to track the space rat's movements and guide its actions toward capturing the target.

The parameter α determines the sensitivity of the rat detection model, controlling how the probability of hearing a detection "ping" decreases with distance. Smaller α values make the sensor more sensitive, allowing it to detect the rat from farther away, while larger α values limit detection to closer distances. However, both extremes of α are less effective: with very small α, the sensor always "pings," providing little useful information, and with very large α, the sensor rarely "pings," offering equally limited insights. The most effective range for α is between 0 and 0.2, where the sensor provides meaningful and actionable feedback, balancing sensitivity and precision.

We plotted several performance metrics against α, including the average number of actions, the average number of rat detection actions, the average number of movements, and average blocked cell sensing actions. These visualizations provided insights into the bot's performance and its ability to effectively track both a stationary and a moving rat.

3.0 Model Overview

1. Input Layer: In our approach, the input layer processes three key components:

- **Rat Probabilities:** A grid indicating the likelihood of the rat's location.
- **Ship Layout:** A grid mapping open paths and obstacles.
- **Bot Position:** A binary grid marking the bot's current location.

These inputs provide a comprehensive view of the grid and problem context for effective decision-making.

2. Processing Through Layers: The convolutional layers analyze spatial patterns to identify pathways and walls. This processed data is then passed to fully connected layers, which transform it into actionable outputs specific to the bot's task of reaching the rat.

3. Output Layer: The output layer predicts a single value representing the estimated number of actions required for the bot to locate the rat. This prediction reflects the model's understanding of the grid and its ability to generate an effective strategy.

4. Significance: This Neural Network architecture enables the model to process environmental data, adapt to changes in the grid, and optimize the bot's movement to effectively locate both static and dynamic rats.

How the Data is Collected?

In **our project**, data is collected by tracking the **bot's movements** and updating its **knowledge** about the environment in real time. At each step, the bot's position is encoded as a **binary grid**, where the cell corresponding to the bot's location is marked as 1, and all others are 0.

Simultaneously, the model tracks the **probability** of the **rat's position** across the grid, storing these probabilities in a **2D array**. These probabilities are updated based on the bot's **distance** from the rat, providing dynamic information about the rat's potential locations.

Additionally, the **ship layout**, which maps **walls** and **open spaces**, is represented as a **binary grid** (1 for walls and 0 for open spaces). This information, along with the rat probabilities and bot position, is combined into an **input tensor**, which is created by stacking these three channels into a **3D array**. The tensor is then added to the **data log**, along with the **timestamp** and **bot's position**, for later use in **model training** or **evaluation**.

Finally, the data collection process is completed when the bot reaches the **rat's position**, indicating that the task has been successfully accomplished. This data is continuously updated as the bot progresses through the grid, helping the model refine its understanding of the environment and improve its **decision-making** ability.

→ **Note**:

This process is carried out for each individual simulation, where the data is continuously recorded at every step. The collected data logs are then aggregated to form a complete dataset, which is used as input for the model.

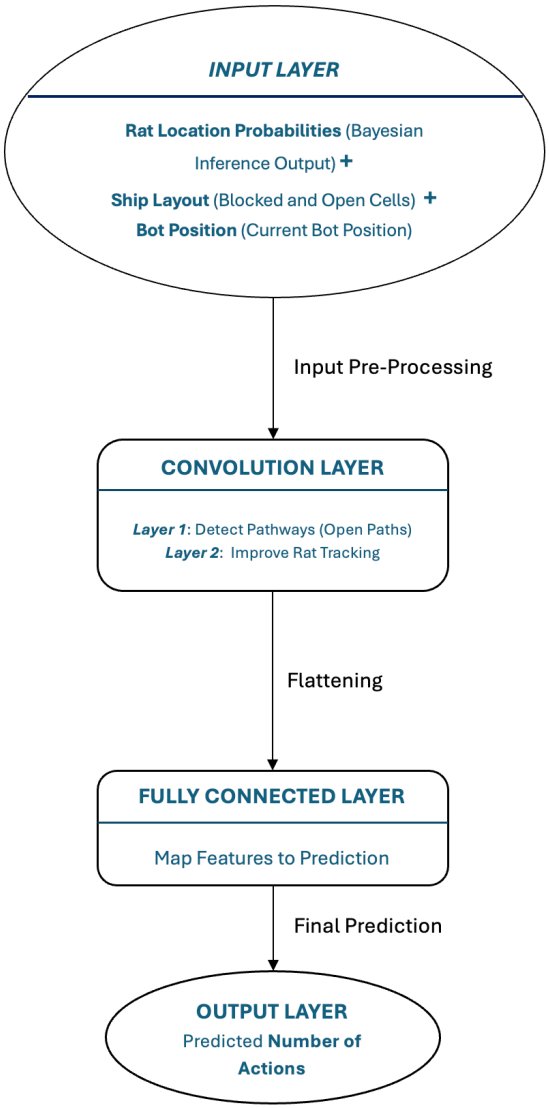

Figure 1: Model Overview

Why Three Input Channels?

Knowledge Base Channel: Represents the probability of the rat's location, guiding the bot's movement strategy toward potential target areas based on the likelihood of the rat's presence.

Ship Layout Channel: Provides the map of the environment, identifying walls and open paths. This channel also accommodates multiple grid permutations, ensuring the model generalizes effectively to varied and complex grids.

Bot Position Channel: Offers an unambiguous understanding of the bot's current location within the environment. It provides:

- **Position Clarity:** Ensures the bot's location is easily distinguishable from other grid features.
- **Dynamic Updates:** Tracks the bot's movements in real-time for adaptive decision-making.
- **Action Planning:** Supports efficient path planning by highlighting the bot's current location relative to the grid and the target.

By integrating these three channels, the model can interpret and process the environment effectively, leading to accurate and reliable predictions for the bot's actions.

4.0 Model Training & Testing

1. Data Collection (Dynamic Inputs for Model Training):

Purpose: Collects input features during the simulation to train and test the model.

Details of Collected Data:

- **Rat Probabilities:**
  → A 2D grid representing the probabilities of the rat being in each cell.
  → Each cell contains a probability value indicating the likelihood of the rat being present there.

- **Ship Layout:**
  → A binary representation of the grid layout:

  `1` *for walls or impassable areas.*
  `0` *for open spaces or walkable areas.*

- **Bot Position:**
  A 2D grid marking the bot's current position with a distinct value, while other cells remain zero.

Input Representation:

→ These three components (rat probabilities, ship layout, and bot position) are combined to form a multi-channel grid.
→ This grid serves as an input representation of the environment for the model to process.

Purpose of Data Logging:

Each collected input, along with associated metadata like timestamps and actions, is stored for later use in training and testing.

2. Training Process:

Purpose: Train the model using the collected data to predict optimal actions, such as steps remaining or the best actions to take.

Steps in Training:

1. **Loss Function:**
   → **Type Used:** Mean Squared Error (MSE), which measures the difference between predicted and actual values.

   → **Equation:**

$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

   $\hat{y}_i$: The predicted value output by the model.
   y_i: The actual target value based on past data.
   N: Total number of samples.

→ **Relevance:**

This loss function ensures the model learns to minimize the error between its predictions and the actual data, improving accuracy over time.

2. **Optimizer:**
   → **Type Used:** Adam, an advanced gradient descent method.

   → **How It Works:** Combines momentum, which smooths updates, and adaptive learning rates, which adjust step sizes dynamically for each parameter.

   → **Advantage:**

   Adam is efficient and robust, especially for noisy or sparse data, making it ideal for dynamic simulation environments.

3. **Training Iterations:**
   → The model is trained over multiple iterations, with the data shuffled for better generalization.
   → In each iteration, predictions are made, errors are calculated using the loss function, and the optimizer updates the model to improve predictions.

4. **Batch Processing:**
   The training data is divided into smaller batches to improve computational efficiency and allow gradient updates after processing each batch.

3. Testing the Model:

Purpose: Evaluate the trained model on unseen data to assess its ability to generalize to new scenarios.

Testing Steps:

1. **Evaluation Mode:**
   The model is switched to evaluation mode to ensure consistent predictions by disabling training-specific features.

2. **Prediction:**
   The model processes the unseen data and generates predictions, which are compared against the actual target values.

3. **Loss Calculation:**
   The test loss is computed using the same loss function (MSE) to measure the model's performance on unseen data.

Significance of Testing:

Comparing training and testing losses helps identify overfitting or underfitting, ensuring the model performs well on real-world data.

Data Analysis:

→ **Note**:

We have chosen an $\alpha = \mathbf{0.1}$, as it provides the best performance by allowing the bot to capture the rat with the fewest actions.

1. Analysis of Training and Testing Loss Graph across Epochs Values (Static Rat):

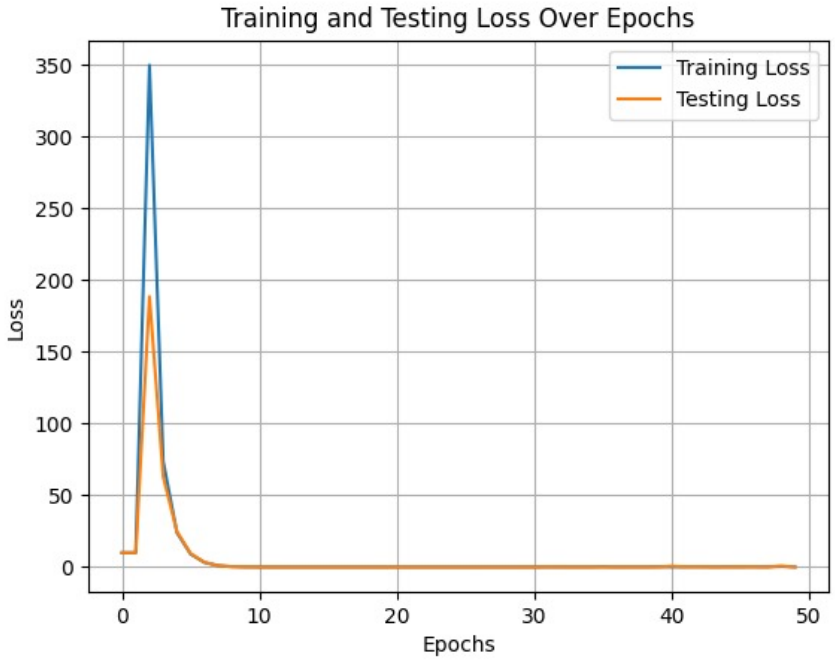

Figure 2: Loss vs. Epochs

→ **Initial Phase (Epochs 1–3):** Training Loss: 349.72 → 73.73, Testing Loss: 188.43 → 62.90.
→ **Middle Phase (Epochs 8–20):** Training Loss: 1.01 → 0.01, Testing Loss: 0.91 → 0.01 (stabilizes around epoch 15).
→ **Later Phase (Epochs 30–50):** Training Loss: 0.02–0.09, Testing Loss: 0.02–0.80 (spikes observed, e.g., 0.80 at epoch 49).

Explanation:

- In the initial phase, losses drop steeply as the model adjusts random weights to capture fundamental spatial relationships in the grid. This sharp reduction highlights the model's ability to adapt quickly in the early stages of training.
- During the middle phase, losses decrease more gradually, and testing loss stabilizes by epoch 15. The close alignment of training and testing losses suggests the model is generalizing effectively, making it well-suited for predicting actions.
- In the later phase, testing loss shows variability, reflecting the model's adaptation to subtle and complex patterns in the data.

2. Specific Analysis of the Loss Graph Across Timestamps (Static Rat):

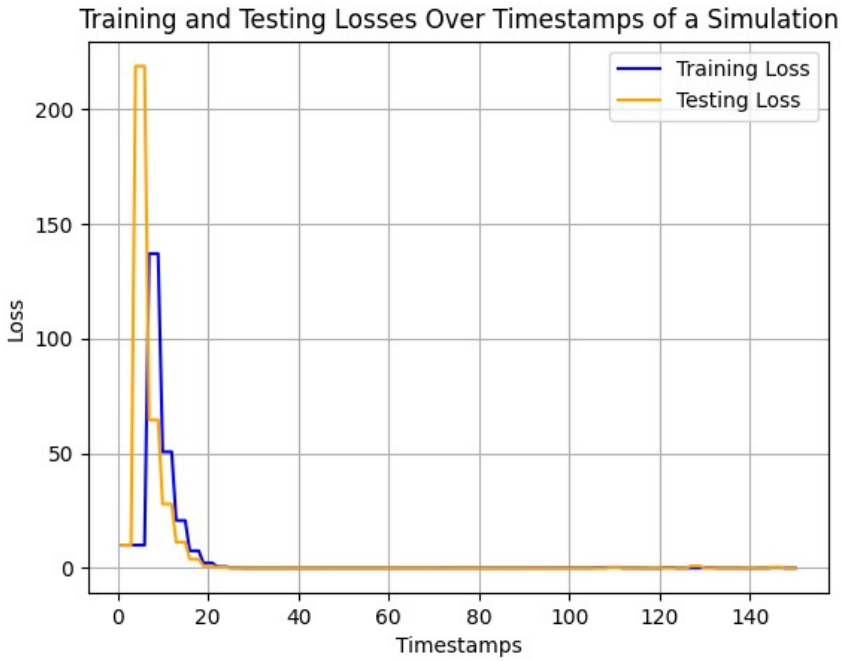

Figure 3: Loss vs. Time Elapsed

This graph represents training and testing losses tracked over timestamps within a single simulation. Unlike the epoch-based graph, this one captures how the model's performance evolves dynamically within the simulation itself:

→ **Initial Phase**: The steep drop in both training and testing losses highlights how the model rapidly learns to adapt to the specific simulation layout and initial conditions. The starting losses are higher due to the initial uncertainty in predictions for this particular scenario.

→ **Stabilization Phase**: Losses flatten quickly, reflecting the model's consistent predictions as it adapts to the simulation environment. This indicates that the model's training effectively generalizes not just across epochs but also across varying conditions within a simulation.

→ **Timestamps-Specific Insight**: Unlike epochs, timestamps provide more detailed information, showing how loss stabilizes in real-time during a specific run. This is particularly useful for assessing how well the model responds to immediate changes in the knowledge base state rather than general trends over training.

T.P.O

3. Analysis of Score Differences Over Time (Static Rat):

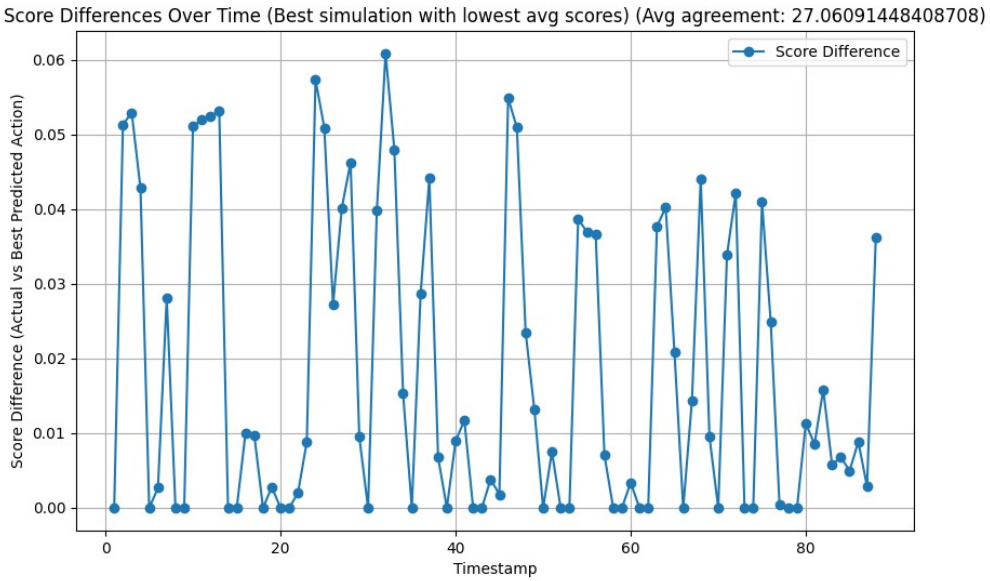

Figure 4: Score Difference vs. Time Elapsed

→ **Spikes**: The spikes in the graph indicate instances where the model predicts a better action with a higher score than the actual action taken by the bot. These highlight points where the model outperforms the bot's decision-making.

→ **Flat Lines at Zero**: When the score difference is 0, it signifies agreement between the model's predicted action and the bot's actual action.

→ **Average Agreement**: The average agreement percentage across 200 simulations illustrates how frequently the model aligns with the bot's actions.

Fluctuations: The graph displays variations with spikes and troughs. Spikes, observed at timestamps such as 10, 30, and 50, indicate significant disagreements where the model strongly suggests alternative actions compared to the bot. In contrast, troughs represent moments of close alignment or agreement between the model's predictions and the bot's actions.

→ **Note**: With a value of 27.06, the average agreement reflects the overall alignment between the model and the bot. While the alignment remains generally consistent, periodic spikes highlight moments where the model identifies opportunities to improve upon the bot's decision-making process. The overall behavior, supported by the training and testing loss trends, confirms that the model is well-trained and capable of making superior decisions in dynamic scenarios.

Score Difference:

Definition: The absolute numerical difference between the action scores of the bot's actual actions and the model's predicted best actions. Lower differences indicate closer agreement, while higher differences suggest disagreement.

Interpretation:

→ A low score difference means a closer alignment with the actual bot's decision. (0 score difference means complete agreement).

→ A high score difference (spikes) suggests the model disagrees significantly, possibly indicating the bot's decision could have been suboptimal.

4. Analysis of Training and Testing Loss Graph across Epochs Values (Dynamic Rat):

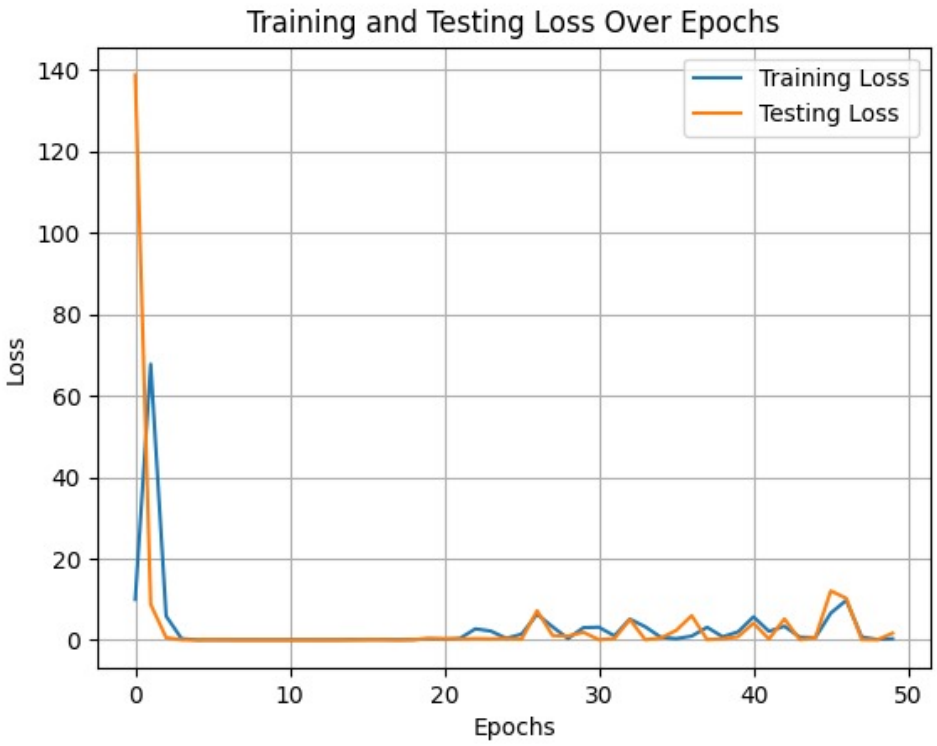

Figure 5: Loss vs. Epochs

Explanation:

→ **Initial Phase:** The rapid drop in losses reflects the model's adaptation to the increased complexity introduced by the dynamic rat's unpredictable movements. This initial adjustment highlights the model's capability to generalize from static patterns to dynamic scenarios.

→ **Middle Phase:** Stabilization of losses around epoch 15 indicates the model's successful learning of dynamic movement patterns, ensuring generalization across simulations involving a rat that changes position at every timestamp.

→ **Later Phase:** In the later phase, the testing loss exhibits minor fluctuations, reflecting the increased variability introduced by the dynamic rat's unpredictable movements. This is expected in dynamic environments, where the model continuously encounters diverse and challenging scenarios, causing slight variations in loss trends.

5. Specific Analysis of the Loss Graph Across Timestamps (Dynamic Rat):

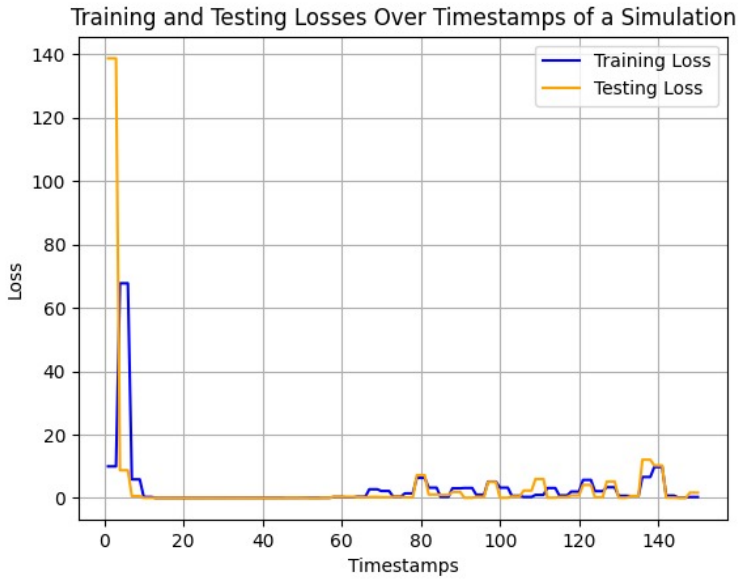

Figure 6: Loss vs. Time Elapsed

→ **Explanation:** In the dynamic rat case, the graph shows an initial steep decline in losses as the model quickly adapts to the simulation. Toward the later timestamps, the testing loss exhibits more pronounced fluctuations compared to the static rat case, likely due to the unpredictability introduced by the rat's movements.

6. Analysis of Score Differences Over Time (Dynamic Rat):

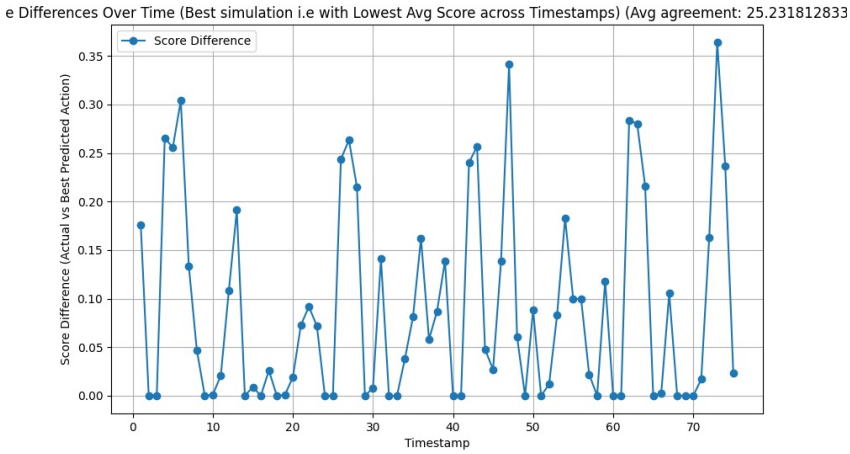

Figure 7: Score Difference vs. Time Elapsed

→ **Explanation:** This graph highlights the Score Difference over time for the dynamic rat case, showing how the model and bot diverge in decision-making. Spikes (e.g., timestamps 10, 30, 70) suggest that the model predicted actions prioritizing long-term navigation success, while the bot may have acted more reactively. These differences emphasize that in dynamic scenarios, such as those requiring continuous adaptation to moving factors or changing layouts, the model's predictions likely reflect strategies learned from broader training, optimizing success probabilities in contrast to the bot's real-time heuristics.

5.0 Predicted vs. Actual Actions (Stretch Goal 1 & 2):

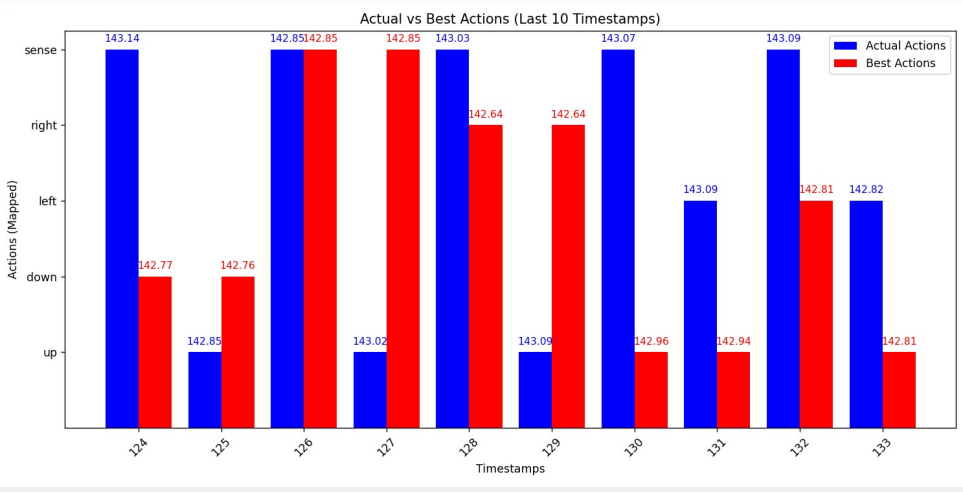

Figure 8: Predicted vs. Actual Actions

Bar Graph (Predicted vs. Actual Actions)- Stretch Goal 1

→ **Note**: The **score** represents the predicted number of actions remaining to catch the rat based on the current state and decision. A **lower score** indicates that fewer actions are needed to achieve the goal, thereby corresponding to a **better action**.

Purpose: Compares actual actions taken during a bot's simulation (blue bars) to the model's predicted actions (red bars).

Key Observations: Timestamps 124 and 125 show clear disagreement between actual actions (blue) and predicted actions (red). This indicates that the model suggests better alternatives to what the bot originally executed. Between timestamps 126 and 130, there is partial or full alignment, showing that either the model agrees with the bot's decision or the actual actions taken were already optimal.

Mapped Actions (Sense, Right, Left, Down, Up): The Actions are denoted across the y-axis. Discrepancies represent differences in decision logic between the bot and the trained model.

High agreement is observed where blue and red bars align (e.g., timestamp 126), confirming that the bot's original actions were effective. Disagreement, where red bars diverge from blue (e.g., timestamps 124 and 125), highlights instances where the model identifies actions that, based on past simulations, might have led to better outcomes.

Why the Actual and Predicted Actions Disagree? Stretch Goal 2

→ **Note**: The peaks in the graph of score difference vs. timestamps depict
where the actual number of actions differs from the predicted number of actions.

1. Model's Learning from Past Simulations

Generalization of Patterns:

- The model has been trained on past simulations, where it observed how different actions impacted the outcomes.
- It has learned generalized patterns and strategies that might improve performance compared to the bot's context-specific decisions.

Potential Improvements:

- The model might predict actions that optimize success metrics (e.g., survival, goal-reaching) better than the bot did during the original simulation.

Example: Timestamp 124:

→ The actual action (blue bar) might represent a suboptimal decision taken during the simulation.
→ The model (red bar) predicts a better action based on what it learned from past scenarios, where similar decisions led to better outcomes.

2. Limited Context Awareness in the Bot

Action Heuristics:

- The bot's actions are often guided by predefined heuristics or rules (e.g., prioritizing immediate safety or moving toward a goal).
- These heuristics might fail in complex situations where a more nuanced decision is needed.

Model's Advantage:

- The model has seen a wide variety of situations during training, allowing it to make decisions that are less rigid and more adaptive to the environment.

Example:

In a scenario with multiple equally viable paths, the bot may take the nearest path (heuristic-based), while the model might predict a path that avoids risks more effectively based on training.

3. Delayed Rewards in Simulations

Short-term vs. Long-term Thinking:

- The bot might prioritize immediate rewards or goals without accounting for long-term consequences.
- The model, trained with simulations involving longer-term outcomes, might predict actions that sacrifice short-term gains for overall success.

Training Signal:

- If the training data emphasizes long-term success, the model will disagree with short-sighted bot actions.

Example:

A bot's action (e.g., moving "up") might avoid immediate danger but lead to a dead-end later. The model predicts "down" instead, foreseeing a safer long-term path.

4. Noise or Suboptimality in Bot's Actions

Human-Like Suboptimality:

- Bots, like humans, can make suboptimal decisions due to incomplete information or poor prioritization.
- Disagreement occurs when the model identifies these flaws and predicts a better alternative.

Learning from Mistakes:

- The model corrects past mistakes made by the bot, resulting in consistent disagreement in similar situations.
- **Example:**

  A bot might sense and move unnecessarily in the same direction repeatedly. The model, recognizing inefficiency, suggests a different action to save energy.

References

[1] Terven, Juan, Cordova-Esparza, Diana-Margarita, Ramirez-Pedraza, Alfonzo, & Chávez Urbiola, Edgar. (2023). *Loss Functions and Metrics in Deep Learning. A Review.* https://doi.org/10.48550/arXiv.2307.02694

[2] Rituraj, Rituraj. (2023). *A Comprehensive Investigation into the Application of Convolutional Neural Networks (ConvNet/CNN) in Smart Grids.* https://doi.org/10.31224/2762

[3] Thanh, Tran. (2021). *Grid Search of Convolutional Neural Network model in the case of load forecasting.* Archives of Electrical Engineering, 70. https://doi.org/10.24425/aee.2021.136050

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* arXiv preprint arXiv:1312.5602.

[5] Rismayanti, Nurul. (2024). *Predicting Online Gaming Behaviour Using Machine Learning Techniques.* Indonesian Journal of Data and Science, 5. https://doi.org/10.56705/ijodas.v5i2.166

[6] Y. Roh, G. Heo, & S. E. Whang. (2021). *A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective.* IEEE Transactions on Knowledge and Data Engineering, 33(4), 1328–1347. https://doi.org/10.1109/TKDE.2020.2990550

[7] Rainio, O., Teuho, J., & Klén, R. (2024). *Evaluation Metrics and Statistical Tests for Machine Learning.* Sci Rep, 14, 6086. https://doi.org/10.1038/s41598-024-62006-w

[8] Maddipati, Himakar, Kundurthi, Aravind, Raaj, Pothula, Srilatha, Kapudasi, & Surapaneni, Ravikishan. (2020). *Artificial Intelligence based Pacman Game.* International Journal of Innovative Technology and Exploring Engineering, 9(140-144). https://doi.org/10.35940/ijitee.I6975.079920

End of Document