# 

# **Modeling Human-Like Learning in AI: Evaluating Recurrent Neural Networks' Capacity to Replicate Human Leaps of Insight in Sequential Decision-Making**

#### 

A Thesis Presented

by

Nina Leah Mizrahi

To the Keck Science Department

of

Claremont McKenna, Scripps, and Pitzer Colleges

In Partial Fulfillment of

The Degree of Bachelor of Arts

Senior Thesis in Computational Neuroscience

December 6th, 2024

#### **Table of Contents**

Abstract.…………………………………………………………………………………………...2

Introduction.……………………………………………………………………………………….4

Literature Review.…………………………………………………………………………...…...12

Methods.………………………………………………………………………………………….17

Results……………………………………………………………………………………………34

Discussion………………………………………………………………………………………..47

Conclusion…………………………………………………………………………………..,…..55

Acknowledgments………………………………………………………………………………..56

Bibliography……………………………………………………………………………………..57

# 

# **Abstract**

This thesis investigates the differences between human and artificial intelligence (AI) learning through gameplay in *Hexxed.io*, a six-sided puzzle game designed to explore problem-solving strategies and learning dynamics. Human learning often involves "leaps of insight"—sudden cognitive shifts to optimal strategies or theories that allow for faster understanding—whereas AI systems rely on incremental, bottom-up processes, refining performance iteratively through repeated exposure to data. This research examines the extent to which RNNs can emulate the top-down dynamics of human learning using bottom-up mechanisms such as gradient descent, gating, and backpropagation, while also analyzing the contribution of non-linearity and input features to model learning.

Recurrent neural networks (RNNs), specifically dense networks with Long Short-Term Memory (LSTM) layers, were trained on gameplay data from 1,647 human players who demonstrated leaps of insight in a controlled game environment. Human action sequences were processed into vectorized representations of board states and previous actions to train multiple RNN variants with varying depths. These models were compared to a linear and non-linear baseline control network. To isolate the contribution of specific features (e.g., board states or previous actions) models were trained on each feature independently.

Results showed that RNNs significantly outperformed the baseline models in capturing temporal and contextual patterns. However, adding more LSTM layers did not improve performance, with the single-layer RNN achieving the highest validation accuracy (68%). These findings demonstrate that temporal dependencies, rather than non-linearity, were the primary drivers of the RNNs’ increased ability to predict human actions, suggesting that temporal dependencies are critical for accurately modeling human-like decision-making.

While RNNs approximated some human behaviors when trained on leap-of-insight data, they failed to replicate the true cognitive shifts characteristic of human learning. Instead, the models relied heavily on patterns more explicitly present in the training data, failing to infer abstract or transferable strategies for handling longer sequences (>200 steps) or those that were underrepresented. These insights point to opportunities for integrating top-down mechanisms (e.g., goal-driven reasoning or reinforcement) into AI systems to enhance human-like AI learning. Future research should attempt to explore hybrid models that combine reinforcement learning with RNNs to incorporate both bottom-up pattern recognition and top-down goal-directed learning.

#### **Introduction**

In the past few years, artificial intelligence (AI) has revolutionized industries and reshaped daily life, firmly establishing itself as one of the greatest scientific advancements of the 21st century. Today, AI can match, if not exceed, the world’s experts in many of the complex tasks that we deem to be benchmarks of human intelligence, such as defeating chess grandmasters and passing the bar exam. This success is in large part due to the fact that most current AI models, especially those used for game play, are built using artificial neural networks—systems inspired by the structure and functioning of the human brain. However, even though AI can learn to perform at a human level over time, it takes these networks far more attempts to learn how to play these games than it does for the average human. For example, DeepMind's AlphaZero taught itself to play chess, shogi, and Go in a matter of hours, outperforming top-ranking masters in these games (Silver et al., 2018). While no human could become a chess grandmaster in just a few hours, this difference is due to AI’s processing speed, not its efficiency. Despite AlphaZero's ability to rapidly process and learn, the computational resources and volume of self-play it requires highlight how much less efficient its learning process is compared to that of humans (Silver et al., 2018). While neural networks excel in all kinds of tasks beyond games, they generally require more training data compared to humans. This suggests that our ability to infer and apply generalizable principles from prior experience is a unique and critical aspect that distinguishes human learning from machine learning algorithms.

##### Bottom-Up (AI) vs. Top-Down (Human) Learning

As Ullman et al. (2012) explore in their paper “Theory learning as stochastic search in the language of thought,” artificial neural networks learn through a process called bottom-up learning, which involves learning solely from raw data, such as patterns, correlations, and reinforcement signals, without any pre-existing knowledge or high-level context to draw from. Learning involves strengthening connections between stimuli (eg. game states) and rewards (eg. accuracy) through mechanisms such as gradient descent, backpropagation and reinforcement learning. While processing speed is very fast, bottom-up learning is inherently slower in that it is iterative, starting from simple connections and gradually becoming more refined with each iteration, as shown in Figure 1B.

In contrast, Ullman et al. (2012) explain that psychology and neuroscience describe human learning as operating in a top-down manner. Humans perceive the world through the lens of theories or narratives that form a “theory space” – a conceptual framework, illustrated in Figure 1A, that helps us make sense of information by organizing it into causally meaningful structures. However, when existing theories prove incorrect or inadequate, humans don’t always shift gradually to a new understanding the way neural networks do. Instead, we often experience sudden, significant shifts in thinking, abandoning our old theories entirely in favor of new ones. These major shifts in understanding –illustrated in Figure 1A as a leap from one peak in the theory space to another– are often referred to as "AHA moments." In this paper, this phenomena will be referred to as leaps of insight. This top-down approach allows humans to integrate broader, high-level theories with finer details, enabling us to refine our understanding at smaller scales in ways that are more adaptable and generalizable to new contexts. What makes human learning so extraordinary is our ability to adapt and solve problems by making these sudden leaps of insight—an ability that artificial intelligence, despite its many rapid advancements, has yet to adequately replicate.



##### *Hexxed.io*

Games are ideal for studying the similarities and differences between human and artificial learning, and can offer insights into flexible and strategic behavior in particular. This is because games are a controlled environment with a defined set of rules and clear, measurable actions, allowing for the systematic observation and analysis of decision-making processes. This research uses an ios-based game, *Hexxed.io*, developed by Professor Gautam Agarwal to investigate strategies and insights demonstrated by players. This game enables us to observe how individuals approach problem-solving in a controlled setting and compare it to AI’s approaches to the same task. In Hexxed, players are presented with a six-sided puzzle without any explanation as to what the objective of the game is or how to win (see Figure 2). They are only given two instructions: they can tap and swipe. They must discover the game’s objectives through experimentation, guided only by directions that they can tap and swipe. On the left of Figure 2, you see how the game is presented to humans—as a six-sided puzzle. On the right is how the game is presented to AI. The same six-sided puzzle is represented as 6 rows of a binary matrix, where 1s mark the target's location. Each of the 6 lobes of the hexagon has 6 segments. To pass the first level, the player must learn to fill up the target lobe by tapping on the target lobe exactly 6 times and then swiping to collect its value.



In the first level alone, players can take over 1,000 possible paths. Across the entire game, there are 164 unique patterns spanning 6 levels, which allows us to test different degrees of difficulty. Since the game is publicly available on iOS, we can gather large-scale data through citizen science. Since we know how the game works, we can train AI agents to play the game and use them as our bottom-up baseline to compare our top-down human learning which takes the form of player action sequences. From previous research which trained and compared reinforcement learning (RL) agents to play the game and compared that to the performance of human subjects, three major trends stood out. These trends can be characterized by the words “picky”,”sticky,” and “leapy” (Quendera et al., 2022). Picky refers to the fact that humans seem to sample only a relatively small subset of the possible actions. Sticky refers to the fact that people prioritize certain action sequences over others. Finally, leapy, the behavior we are interested in here, is when human participants may suddenly and unpredictably arrive at an optimal solution, called a "leap of insight." This leapy behavior is visualized well in Figure 3, a graph comparing the progression of human and artificial learning across attempts of level 1.



Each line represents the learning of an individual human or artificial agent. On the x-axis is the number of attempts, on the y-axis is progress where the middle line represents the threshold at which they have learned or completed the puzzle. Looking at the progression of RL agents, represented in red, we can see that they will gradually increase in performance until eventually they will stumble into some kind of solution. As expected, the ascent to understanding is markedly different. Humans tend to remain stuck at relatively low levels of performance until at some point they suddenly spike and pass the level. This is what we call a leap of insight.

##### Objectives and Scope

This research examines the extent to which Recurrent Neural Networks (RNNs) can emulate the top-down dynamics of human learning using bottom-up mechanisms such as gradient descent, gating, and backpropagation, while also analyzing the contribution of non-linearity and input features to model learning. Our research aims to tackle a notoriously difficult and underexplored problem: capturing the spontaneity, creativity, and intuition involved in human problem-solving. While AI, particularly RL agents, are excellent at optimizing performance, they lack the spontaneous creativity and adaptability of human problem-solving. This comes at the cost of extensive and costly training processes. Current AI models, which are rooted in gradient descent, are inherently iterative and struggle to replicate leaps of insight. Human learning and decision-making are often sequential processes; our actions depend on our current state, past actions, and the outcomes of those actions. Recurrent neural networks (RNNs) are designed specifically to model such sequential dependencies. They can “remember” prior steps, which is critical for capturing patterns in behavior. Therefore, the task for our models uses a classic sequence modeling problem where we have our model predict the next action given the current game state. RNNs work well when the output (e.g., next action) is tightly tied to the sequence of inputs (e.g., board states + prior actions). Long-short-term-memory (LSTM) layers are a type of RNN architecture particularly good at handling long-term dependencies (e.g., when decisions made early in a sequence influence behavior later). This makes them a logical choice for studying nuanced behaviors like leaps of insight, which require context over time.

This thesis aims to explore how neural networks, specifically RNNs, can interpret and approximate human problem-solving strategies, including leaps of insight. This goal can be broken up into the following key research questions:

1. To what extent can RNNs, using bottom-up mechanisms like gradient descent, gating, and backpropagation, emulate the top-down dynamics of human learning, including strategic flexibility and sudden insights demonstrated by human players in *Hexxed.io*?
2. How well do RNNs, with their ability to handle sequential dependencies, perform in modeling strategy shifts and human behavior compared to baseline models?
3. How does the addition of recurrent layers influence the RNN’s ability to capture and replicate human-like learning behaviors?
4. What impact do non-linearity and specific input features have on the model's ability to generalize and predict player actions during learning.

We hypothesize that an RNN model, with its ability to process sequential data and maintain long-term dependencies, will more accurately model the leap-like strategy shifts observed in human players compared to baseline models. As we add recurrent layers to the network, we expect it to better capture these shifts, with improved accuracy in the RNN's predictions, particularly for steps later in the sequence. Additionally, we hypothesize that leaps of insight are directly informed by a player's previous actions and current board state. Therefore, across all models, we expect that the combined gamestate (previous action and current board state) will serve as a better predictor of player behavior compared to each feature trained in isolation. More specifically, we predict that our RNN models will be able to analyze relationships between game states and actions, compressing information across states to predict future strategic movements. Ultimately, the model will capture player distribution statistics such as state durations and transition probabilities, representing the underlying dynamics of human learning.

Though we will be relying on the accuracy of the model to predict the next action in a sequence as an evaluation metric, the goal of this study is not to build a perfect model, it is to explore how well existing tools can replicate human learning behaviors and uncover their limitations. Demonstrating the ability of bottom-up mechanisms to capture the top-down dynamics of human learning and strategy shifts would be an important breakthrough. Current understanding suggests that gradient descent learning, the AI industry standard, fundamentally contrasts with leaps of insight. This idea suggests that although leaps of insight are commonly associated with top-down processing, they can also emerge from bottom-up processes like gradient descent. If so, iterative approaches like gradient descent could occasionally yield unexpected breakthroughs, bridging the gap between human-like cognitive processes and AI learning mechanisms. Such findings could lead to notable advancements in acquired strategies and solutions, particularly in areas like game design and behavioral prediction, where understanding and replicating human learning behaviors are critical.

# **Literature Review and Theoretical foundations**

The purpose of this literature review is to contextualize the creation and findings of our research and explain the rationale behind selecting LSTMs as the primary model for studying sequential dependencies and replicating human learning dynamics.

##### Theoretical foundations: Comparison of Human and AI Learning

As mentioned earlier, the comparison between human and AI learning is often characterized as top-down vs bottom-up, respectively. In his paper "*Bottom-Up Learning and Top-Down Learning in cognitive skill acquisition*," Ron Sun defines top-down learning similarly to Ullman et al., stating that it involves the acquisition of implicit knowledge first, followed by explicit knowledge. This learning is primarily unsupervised, meaning it involves the integration of sensory experiences, context, and prior knowledge to understand and interact with the world. (Sun, 2004). In contrast, Sun defines bottom-up learning as the acquisition of explicit knowledge before implicit knowledge. Any type of “perception” that today's AI models develop during training is passive and data-driven in the sense that it predominantly relies on supervised learning, where the model is trained on large amounts of labeled data from which it recognizes patterns and makes predictions (Fodor, 2022; Slagter, 2023). As such, these models can only generalize what they learn from training data to new, similar inputs, and often struggle with tasks which require contextual understanding and adaptation to novel situations without the need to be retrained (Fodor, 2022). This phenomenon, known as “transfer learning” is a critical area of ongoing research and highlights the limitations of bottom-up approaches which often lack creativity and transferability of ideas. For this reason, it comes as no surprise that recent research suggests that transfer learning can be enhanced with the introduction of top-down mechanisms and the ability to connect to higher levels in the processing hierarchy (Slagter, 2023).

##### Imitation learning

One way implementation of these mechanisms is being explored is through imitation learning, where AI systems learn by observing and replicating the actions of human agents. The goal of Imitation learning is to observe and replicate successful behaviors, allowing for the faster acquisition of new skills and improving performance across diverse tasks. By training on and imitating successful human demonstrations, often in a supervised or semi-supervised manner, these agents can acquire complex skills or complete tasks more quickly and efficiently. This technique also reduces the burden of manually programming explicit instructions for every task, making it a valuable tool for developing more flexible and intelligent systems (Ahammed, 2024). This study aims to apply this same approach to our models by training them on data from Hexxed players who have demonstrated a leap of insight, enabling the models to learn and replicate the spontaneous strategy shifts observed in human gameplay. This approach was similarly successful in the paper *"Integrated Machine Learning for Behavior Modeling in Video Games,"* where author Ben Geisler demonstrated that training models on data collected from skilled players enabled AI agents to exhibit more lifelike and unpredictable behaviors, significantly enhancing the gaming experience. Additionally, as highlighted in a recent MIT study on modeling irrational behavior, incorporating suboptimal or unexpected actions into the training process actually enables AI to better anticipate and adapt to real-world human decision-making (Zewe, 2024).

Another facet of imitation learning is that, in humans, perception and action co-develop over time (Gibson, 1988). During the learning process, humans continuously integrate sensory information with motor activities to interact effectively with our environment. This interplay allows the discovery of new actions and possibilities within an environment, which in turn guides further exploration and learning. In the context of the AI models in this study, incorporating both the current board state (perception) and the previous action (action) aims to mirror this human learning process. By providing the model with information about the current state of the game and the action that led to it, we will hopefully enable our models to capture patterns of complex high-level reasoning and understand the consequences of specific actions within particular contexts.

This brings us to one of the main disparities between human and artificial learning which is the integration of context and experience. AI lacks consciousness and does not possess an inherent understanding of the information it processes; it operates based on statistical correlations rather than comprehension. In contrast, human cognition involves awareness, intentionality, and the ability to infer meaning, enabling deeper understanding and reasoning (Fodor, 2022). Therefore, while AI’s working memory can handle extraordinary amounts of data, it lacks a mechanism that parallels human’s ability to integrate experiences and context. This disparity underscores the importance of selecting models that can attempt to replicate, at least partially, the sequential and contextual processing inherent to human cognition. RNNs were chosen for our research because they offer a mechanism that does just this.

##### RNNs and LSTMs

What makes RNNs different from regular neural networks is their ability to "remember" information from previous steps in the sequence, which helps them make better predictions for the current step. As visualized in Figure 4, the output of the recurrent layer at each step is influenced by its previous outputs, creating a loop that helps the network remember and use past information to inform future predictions (Graves, 2012). Human learning and decision-making are often sequential processes where actions depend on the current state, past actions, and the outcomes of those actions. RNNs (and their improved variants like LSTMs) are designed specifically to model long-term sequential dependencies (e.g., when decisions made early in a sequence influence behavior later) and have demonstrated significant success in using that ability to predict human-like behavior based on sequential data (Ubal, 2023; Sharma, 2023). This makes them a logical choice for studying nuanced behaviors like leaps of insight, which require context over time. Additionally, they have been shown to outperform other models in predicting strategically informed sequential behavior, such as that observed in test-taking scenarios (Tang, S., & Li, Z., 2023; Conroy et. al, 2024).



To further justify our choice of RNNs, it is important to consider why alternative approaches, such as reinforcement learning (RL), were not selected for this study. RL is most effective when the goal is learning how to act optimally in an environment based on trial-and-error feedback (rewards and penalties). In the Hexxed environment, this may be useful in having an AI learn the best strategy to solve the puzzle itself, but it doesn’t inherently focus on predicting human actions or strategies. Our goal is to understand and replicate how humans learn and strategize, not necessarily to create an AI that solves the game perfectly. RNNs are better suited for modeling behavioral sequences because they explicitly predict what comes next in a sequence, rather than optimizing actions to maximize rewards. Additionally, while other models such as Transformers are highly effective for sequence tasks, they are often excessive for small-scale problems like ours (Vaswani, 2017). RNNs/LSTMs strike a balance between simplicity and power for our specific task and dataset.

# **Methods**

##### Overview

This experiment uses data gathered from Hexxed players who demonstrated learning to train both generalized linear dense networks and recurrent neural networks (RNNs). Three experimental models will be implemented: dense RNNs with varying numbers of hidden LSTM layers (1-3). Our baseline control model will be a dense network with no hidden layers and a soft max output. This effectively acts as a linear classifier which tells us how much the RNN ability to handle sequential dependencies has an impact on the accuracy of prediction. Additionally, a second baseline model will be constructed to include an extra hidden dense layer. This allows us to differentiate the performance improvements due to LSTM layers from those caused by merely adding a hidden layer. Both dense networks treat each action independently, ignoring sequential relationships between features and targets. Together, these baselines will help attribute performance improvements in the RNNs to their ability to capture dynamic strategy changes.

Our dataset includes action sequences from 1,647 players who successfully completed the level without explicit instructions, other than being told they can tap and swipe. Each puzzle features a hexagon with six lobes (as shown in Figure 2), with the target lobe marked by a grey bar. To pass level one (wave 1), players must learn to fill the target lobe (e.g., by tapping it six times) and then collect its contents by swiping. This sequence must be repeated six times to complete the level. Once the lobes have been swiped or filled beyond 6, the puzzle board resets, marking a new sublevel (subwave) attempt. Since the correct sequence must be repeated multiple times to win (i.e. has to be repeated across six subwaves), we infer that these players experienced a leap of insight into how the game works, rather than stumbling upon a solution by accident.

The general framework of this experiment is as follows. The actions of a single user are strung together in an action sequence. These are then book marked in a single subwave attempt. When these subwave sequences are strung together, they form a timeline of learning that reveals the progression of behavioral strategies, modifications to those strategies, and, ultimately, the emergence of leaps of insight. The addition of the corresponding board state to each of the actions (creating the game state) introduces multimodal contextual information.

##### Actions and action sequence construction

The "action\_sequences" column, which serves as our target, is constructed of individual user actions during puzzle-solving, which are tracked through the mo\_y, mo\_x, mo\_fake\_release, and similar columns which capture movement and interaction metrics such as touch and release. Based on the location of taps and drags (filtered to be only within the hexagon), it determines what kind of action the action is (tap or swipe) and where the action is being applied in the environment based on the lobe index. These actions are processed and organized into sequences for each user (ID) and session (session\_nr) so that we have movement/interaction metrics for the unique players ID at every subwave. Actions are normalized in terms of their proximity to the target by calculating the relative position between the lobe action takes place in and the target location. Normalized actions are given a value between 0 and 11 indicating where their normalized location is. 0 through 5 are taps and 6 through 11 representing swipes. For each player ID, the total number of actions from each subwave attempt are collected chronologically and stored in a list which represents their action sequence.

##### Preprocessing

The goal of preprocessing was to generate a per-user sequence of game states (features) and their corresponding next action (targets). To represent the six-sided board state to our network, we utilized a pre-existing Hexxed Gym environment developed by BeatLab for training artificial agents. This environment simulates board states, actions, and rewards, allowing us to generate data that captures the relationship between player actions and board configurations. Within this environment, the board state is encoded as a 6x12 (72-element) matrix, illustrated in Figure 2 above. The first 6x6 portion of the matrix represents the grey hint bar, which provides guidance to the player by highlighting the target area before the action is taken. This hint is removed after the first action, as seen in the second image of Figure 2. The second 6x6 portion represents the target area, where the player’s action location is one-hot encoded relative to the target.

Within this environment, each board state is determined by a combination of the player's actions and the environment’s predefined parameters, including grid size, number of levels, and pattern order. The initialization parameters (e.g., num\_vertices, step\_per\_pattern, levels, and shuffle\_patterns) establish the complexity of the game, while the action space, defined as spaces. Discrete(num\_vertices + 1), represents possible moves within the hexagonal structure of the game grid. During gameplay, the environment’s step function processes each user action, updates the board grid with the new state.

To generate and collect board states corresponding to user actions, we implemented a custom function, get\_board\_states\_and\_positions. This function takes a player's action sequence as input and collects the initial state of the board, followed by generating the board state for each action in the sequence except the last which does not have an associated target. For each action in the sequence, the function:

1. Resets the environment using the reset() function to ensure a clean slate for each subwave attempt and records the normalized initial state of the board before any actions are taken. Note that initial states are normalized so that the actions of the player are represented the same for all.
2. Applies the current action to the previous board state (or the initial board state for the first action), updating the game grid to reflect the player’s move.
3. Encodes the current action as a one-hot vector of size 13, appending it to the updated board state. This results in an 85-element vector that represents both the board state and the associated action for each step in the sequence.

Each action modifies the grid’s configuration, altering the game’s state in accordance with user action. The function iteratively processes the entire action sequence, creating a list of game states for each action. The output board states, combined with one-hot encoded positions, are then stored and analyzed to reflect the impact of user actions across sequences. Note that since the initial board state does not have an associated action, its one-hot encoding is represented as [1,0,0,0,0,0,0,0,0,0,0,0,0], while subsequent actions are encoded with a 1 at the appropriate position corresponding to the action. By combining the game state (board state) and action (one-hot encoded vector), the resulting dataset provides a representation that models the relationship between user actions and the evolving game state which is essential for training and validating the computational model in subsequent analyses. At the end of this process, each player's action sequence is associated with a complete list of board states that captures their progress throughout a subwave attempt.

Action sequences record players’ actions for each gameplay session of Hexxed. If a player closes the game and resumes later, this is recorded as a separate row. However, to analyze a player’s complete strategy and learning history for level one, the dataset was processed to represent each unique player ID exactly once. This involved first counting the rows associated with each player ID to identify duplicates. Then, the dataset was then grouped by ID and sorted chronologically by the datetime column to maintain the correct order of actions in each player's history. Rows with unique IDs were isolated into a separate DataFrame, preserving only the necessary columns (ID, action\_sequences, and board\_states). For rows with duplicate IDs, the action\_sequences and board\_states—both stored as lists—were concatenated for each duplicate ID group using a summation operation, combining the sequences and board states into a single entry per ID. Finally, the two dataframes were merged back together, ensuring that each player ID was represented by a single row which retained the complete history of each their interactions with the game.

Due to frequent RAM crashes during processing, the distribution of action sequence lengths were analyzed to determine a truncation point for the dataset (see Figure 5). To retain a broader range of data while addressing memory limitations, I chose to truncate the dataset at the third-longest sequence length, 704, which falls well within the 99th percentile of sequence lengths. This decision was made to optimize memory usage while preserving the majority of the dataset's variability for analysis.



##### Negative Control models

Negative control models (NCMs) are useful for isolating certain variables to assess the true effect or performance of the experimental and control models. The first NCM employed was a random prediction model which outputs actions based solely on a uniform probability distribution. This model generates random predictions with no learning or adaptation to input data, providing a baseline to compare the model's accuracy against what would be expected by chance. Since there are 12 actions we would expect this value to be 1/12 or roughly 8.33%.

The second NCM created was a static prediction model. This model operates by predicting a single, constant action for all instances in the dataset, regardless of the input. Specifically, for each possible action (e.g., actions 0 through 11), the model makes a constant prediction and calculates its accuracy by comparing these predictions to the actual actions taken. The static prediction model serves two key purposes. First, it provides a baseline to evaluate whether more complex models, such as dense networks or RNNs, offer meaningful improvements. If these advanced models fail to outperform the static prediction model, it suggests that they may not be learning meaningful patterns from the data and are just learning to predict the most common action. Second, the model offers insights into the dataset's class distribution since the accuracy of a constant prediction depends on how frequently each action occurs. For example, if a particular action is highly common in the dataset, predicting it consistently might yield a relatively high accuracy, reflecting class imbalance rather than shortcoming of predictive capability.

##### Control networks

To establish a baseline model for comparison, a multinomial logistic regression model, in the form of a dense neural network model with no hidden layers, was constructed. This model acts as an essential control for evaluating the performance of the RNN. As a linear model, it provides a straightforward benchmark: if the RNN outperforms this dense network, the improvement can be attributed to the RNN's ability to handle sequential data and capture dynamic behaviors. The linear model, due to its simplicity, required us to flatten the 'action\_sequences' and 'board\_states' columns so that each sequence value in a row corresponds to a single action and its corresponding game state. This process inherently removed the sequential aspect of the data, making the model a useful tool for assessing the benefit of more complex, sequential models. The architecture of the model consisted of a single output layer with a softmax activation function to predict one of the available actions based on the input game state (Figure 6).



Softmax was chosen because it converts the output into probabilities (Pj), allowing the model to represent the likelihood of each possible action being the correct choice, making it ideal for multi-class classification problems. For a given sample, the softmax is computed as:

Where Zj is the raw score (logit) for class j and C is equal to the total number of classes. The model was compiled with a sparse categorical cross-entropy loss function, which works directly with integer-encoded class labels, eliminating the need for one-hot encoding of our targets. The loss is calculated using the negative log probability of the true class (Pyi), which increases as the model's confidence in the correct class decreases.

)

Combined with the softmax activation, this loss function penalizes predictions that assign lower probabilities to the correct class, encouraging the model to align its probabilities with the true labels:

)

By minimizing this loss, the model is encouraged to assign higher probabilities to the correct classes, improving its predictive accuracy.

An Adam optimizer with a learning rate of 0.0001 was applied. For this and all other models mentioned in this paper, learning rate alongside number of units per hidden layer, were found through an Optuna parameter search (Optuna: Akiba et al., 2019). Learning rate was optimized by testing values on a logarithmic scale, ranging from 1×10−6  to 1×10−1. The size of the hidden layers was explored by evaluating unit sizes in powers of two, from 23 (8 units) to 28 (256 units). Each parameter search was run for 50 trials, with models trained for up to 50 epochs and a batch size of 32. These parameters were kept consistent across all searches. Early stopping with a patience of 10 epochs was applied to prevent overfitting during training. The best-performing configurations were selected based on the highest validation accuracy achieved during the trials.

To train our first model, the columns were then split into training and validation sets, with 20% allocated to validation, using a random seed of 42 to maintain consistency. The feature and target rows of our dataframe were split before being flattened to maintain the integrity of the sequential data by ensuring that no individual sequence was divided between the training and validation sets.The model was trained for up to 50 epochs, with early stopping applied to monitor validation accuracy, using a patience of 10 epochs. This approach helped prevent overfitting and ensured the model retained the weights from the epoch with the best validation accuracy. Batch size was set to 32 for training, and the final performance metrics—loss and accuracy—were recorded for both the training and validation sets.

To further isolate variables that might impact player decision-making, the control model was reconstructed and modified in three key ways. Firstly, the model was modified to train on only the previous action to predict the next action. This was done by slicing each feature to get only the last 13 elements of each game state, the one-hot-encoded previous action. All parameters and hyperparameters were kept the same as the original except for the size of the dense layer input which was changed from (85,) to (13,). The second reconstruction of my control was very similar to the last but was modified to train on only the board state to predict the next action. This was done by slicing features to get the first 72 elements of each game state, and changing the input layer from (85,) to (72,). The third and final reconstruction of my control is exactly the same as the original but includes an additional hidden layer (see Figure 6). The hidden layer consists of 64 neurons and a ReLU activation function. The number of neurons was chosen through Optuna parameter search and a ReLU activation function is used because it introduces non-linearity to the model, allowing it to capture complex patterns in the data. Similarly to the original control, this model has a dense layer input size of (85,) and an and a output layer consisting of num\_actions neurons (equal to 12, the number of possible actions) meaning it aims to train gamestate to predict the next action. The addition of the hidden layer provided a direct comparison with the RNN models, helping to distinguish the effects of sequential dependencies from improvements due to increased model complexity. By isolating these factors, we can better understand the contributions of different model architectures to the overall predictive accuracy.

##### RNNs

The RNNs use the same basic structure as the control model with a dense output layer with 12 output units, corresponding to the total number of possible actions in the dataset and a softmax activation function was applied to output a probability distribution over the 12 action classes for each time step. However, to handle the sequential nature of a players history in which they develop their strategy, we chose to implement recursive network layers. Specifically, we chose to use LSTM layers because of their ability to handle long-term dependencies and retain information across longer sequences. They address the limitations of traditional RNNs, which struggle to handle long-term dependencies, by incorporating a mechanism that controls the flow of information through a series of gates within each cell. These gates enable the network to focus on the most relevant aspects of the input sequence at each timestep by regulating which information is retained, updated, or discarded. An LSTM cell consists of three main gates: the forget gate, the input gate, and the output gate. The forget gate decides which parts of the previous cell state should be removed by applying a sigmoid activation function to the previous hidden state and the current input:

Here, the current game state (xt) and the hidden state from the previous timestep (xt-1), which contains information about past predictions and sequences, are multiplied by the weights of the forget gate (Wf) and added to the bias for the forget gate (bf). They are then passed through the sigmoid activation function, which outputs a value between 0 and 1, where 0 means "forget" and 1 means "retain." This value (ft) is applied directly to the previous cell state (Ct-1) to scale it, determining how much of the past information should be preserved in the current cell state (Ct). The input gate determines what new information (it) should be added to the cell state, combining a sigmoid function to locate updates and a tanh function to generate candidate values () for updating the current cell state:

The product it⋅ determines how much new information to incorporate. Together with the scaled output of the forget gate (ft⋅Ct-1​), these updates create the new cell state:

The output gate controls what information from the updated cell state (Ct) is passed to the hidden state (ht), which is used for the model’s predictions.

The cell state is transformed by a tanh function, tanh(Ct​), to scale it between -1 and 1, and the output gate scales this with ot​:

This hidden state, ht​, captures the key information about the sequence up to the current timestep and is passed to both the next timestep and the output layer for predicting the next action. This entire process is visualized in Figure 7 below.



At each timestep, the LSTM processes the current input, the previous hidden state, and the previous cell state. This design allows LSTMs to retain long-term dependencies and selectively forget irrelevant details, making them effective for sequential tasks.

To be able to be processed by LSTM layers, sequences need to be discrete and three-dimensional. As such we needed to reformat the input data so that the model would be able to identify user sequences and capture the temporal relationships between their actions. This involved restructuring the input data into a 3D format of shape (number of sequences, sequence length, feature size), where each sequence represented a player's series of actions and board states over time. However, since different players have different length sequences, we created a padded tensor for both the input and target data to standardize the input dimensions for our RNN. To ensure uniform sequence lengths, all board states were padded to the maximum length of 704 actions. For shorter sequences, padding was applied by appending vectors filled with a placeholder value of -1. This value was chosen to signify padding and distinguish it from actual data. The padded sequences were then stored in the dataset as a new column, padded\_sequences, and converted into a NumPy tensor for model training. Action sequences, which represent the player’s selected actions during gameplay, were padded similarly. Since the possible actions are represented by integers from 0-11, a placeholder value of 12 was used to pad shorter sequences. This ensured that padding could be masked effectively during training, preventing it from influencing model performance.

To enable the model to recognize padding values without mistaking them for valid actions or features, I implemented custom loss and accuracy functions that explicitly mask the padded elements during computation. This ensures that only actual data points are used for evaluating model performance, preventing the model from being penalized for predictions on padded values. The loss function, masked\_sparse\_categorical\_crossentropy, was designed to ignore padding elements of targets (set to 12) when calculating loss. The function first creates a boolean mask that identifies positions in y\_true that are not equal to the padding value. This mask flags the valid data points. Then, all padding values in y\_true are replaced with 0, ensuring that padding elements do not contribute to the loss calculation and avoiding errors from the 12s being invalid actions in num\_actions. Next, sparse categorical cross-entropy is computed between the modified y\_true (with padding values replaced) and y\_pred as it was for the control models. The resulting loss is element-wise multiplied by the mask to ignore contributions from the padded elements. Finally, the sum of the masked loss is divided by the number of non-padded elements (tf.reduce\_sum(mask)) to compute the final loss. A small epsilon is added to the denominator to avoid division by zero.

The accuracy metric, masked\_accuracy, was similarly adapted to ignore padding elements during evaluation. The predicted action class is obtained using tf.argmax(y\_pred, axis=-1). These predicted classes are compared to the true values, producing a boolean tensor of correct predictions. This tensor is then multiplied by the mask, ensuring only valid data points contribute to the accuracy calculation. The sum of the masked correct predictions is divided by the number of non-padded elements to compute the final accuracy.

In terms of their architecture, the three RNNs differ only in their number of hidden LSTM layers (See Figure 8). Each model begins with a masking layer with an input shape of (None, 85). This masking layer ignores padding values (-1) in the input sequences, ensuring that padded timesteps introduced during preprocessing do not influence the learning process. The masking layer informs subsequent layers to disregard these padded elements. Each RNN in this study includes at least one LSTM layer: RNN1 contains a single LSTM layer, RNN2 is configured with two LSTM layers, and RNN3 incorporates three LSTM layers. For all models, each LSTM layer consists of 64 units with a tanh activation function. The tanh activation function outputs values in the range [-1, 1], allowing the model to capture both increases and decreases relative to a baseline. This property is particularly advantageous for sequential data because it allows the model to represent meaningful deviations, such as movements toward or away from a target. This helps the model learn patterns not only in the presence of features but also in their relative direction or magnitude, which is critical for predicting complex behaviors in sequential data. The LSTM layers form the core of the model, capturing temporal relationships within the sequence data. The return\_sequences=True parameter ensures that the LSTM outputs predictions for each timestep, enabling multi-step predictions. Each RNN ends with a dense output layer containing 12 units and a softmax activation, corresponding to the total number of possible actions in the dataset. The model was compiled using the Adam optimizer with a learning rate of 0.005, a custom masked sparse categorical cross-entropy loss to ignore padding, and a masked accuracy metric to evaluate performance on valid data points. Both the learning rate and the number of units per LSTM layer was the same for all three models. Additionally, each RNN was trained on padded input and target tensors using a batch size of 32 for up to 100 epochs, with an 80/20 split for training and validation. Early stopping was applied to halt training if validation accuracy did not improve for 10 consecutive epochs, and the best weights were restored.



As was done with the control model, two supplemental RNN models were to isolate the impact of board state and previous action alone. a slice\_and\_pad\_board\_states function was created which works the same as the padding function described above but first slices the first 72 or last 13 elements of each 85-element board state vector in the sequence before padding. The data is then split.

##### Sequence tracking

The final step in our experiment was to implement sequence tracking for our models to be able to visualize and analyze how players' choices depend on their history. To analyze how accuracy varied across different parts of a sequence, performance was tracked based on sequence position and length. For each model prediction, we compared the predicted action to the actual action and calculated accuracy at each position:

Both regular sequence positions (steps from the start of the sequence) and reversed positions (steps from the end of the sequence) were used to gain insights into how models performed at different points within a sequence. I grouped predictions by position and calculated average accuracy for each group, which revealed trends across sequence positions. To simplify analysis, I also binned sequence positions into ranges (e.g., 0-1, 2-5) and computed average accuracy within each bin. Additionally, I examined performance across entire sequences by grouping accuracy data by sequence length, excluding padded steps to ensure fairness.

# **Results**

The random prediction model resulted in a training accuracy of 0.0837 and a validation accuracy of 0.0830. These roughly equate to 1/12 which is expected given that there are 12 possible actions. The results of the static prediction model are shown in Figure 9 below. The action “0,” representing a tap on the target, was the most common and achieved the highest accuracy of 0.4746. Therefore, any model accuracy exceeding this value indicates the model is identifying meaningful patterns in the data rather than relying on chance or the frequency of a single action.



Below are the validation losses and accuracies for each model trained on gamestate, along with their plotted training history. Note that the values in the table were calculated using the Keras .evaluate method, which aggregates results over the entire validation set. This is largely considered a more reliable measure of accuracy and loss because it computes the loss and metrics on the full validation dataset after training. In contrast, the validation accuracy plotted is slightly different because it is calculated batch-by-batch during training. Therefore it can fluctuate depending on the subset of validation data seen in each epoch and is a less consistent indicator of the model’s performance (Chollet & contributors, n.d.).

| **Model** | **Performance metric** | **Game state** | **Previous action only** | **Board state only** |
| --- | --- | --- | --- | --- |
| Control | loss | 1.2445 | 1.4238 | 1.4215 |
|  | accuracy | 0.6367 | 0.5856 | 0.5748 |
| Control with hidden layer | loss | 1.2186 | 1.4103 | 1.4218 |
|  | accuracy | 0.6415 | 0.5858 | 0.5748 |
| RNN1 | loss | **1.0491** | 1.0846 | 1.1379 |
|  | accuracy | **0.6798** | 0.6664 | 0.6542 |
| RNN2 | loss | 1.052 | 1.0833 | 1.1287 |
|  | accuracy | 0.6788 | 0.6676 | 0.6563 |
| RNN3 | loss | 1.0626 | 1.0873 | 1.1384 |
|  | accuracy | 0.6773 | 0.6701 | 0.6515 |

****

As shown in Figure 10, all models performed significantly better than the random prediction and static prediction negative control models, indicating that they captured meaningful patterns in the data rather than relying on the dominance of the most frequent action or random chance. All three RNN models achieved lower validation losses (~1.05–1.06) compared to the control models (~1.22–1.24) indicating better predictive performance. Additionally, the RNN models achieved higher validation accuracies (~0.677–0.680) compared to the control models (~0.636–0.641), demonstrating the RNNs' ability to capture temporal patterns in the sequence data. The training accuracies and losses support our hypothesis that the addition of LSTM layers will improve the models performance. Interestingly, this trend does not persist for the validation data with the RNN1 performing best and RNN2 and RNN3 performing similarly but slightly worse.

Adding a hidden dense layer to our basic control model showed a slight improvement in performance, with a slight increase in accuracy (~0.005) and a decrease in validation loss (~0.02) compared to the basic control model. However, RNN1 with one hidden LSTM layer still performed significantly better than both control models. It achieved the lowest validation loss (1.049) and the highest validation accuracy (0.680). Compared to the dense model with one hidden layer, RNN1 achieved a 13.9% lower validation loss and a 6.0% higher validation accuracy. When compared to the baseline control model, RNN1 improved the validation loss by 15.7% and the validation accuracy by 6.8% (see Figure 10 and 11).



Visualizing the biases of the baseline control model after training reveals a pattern closely resembling the accuracy distribution of the static prediction model. The most common actions have positive bias values, while the less common actions have negative values (see Figure 12). This indicates that the control model with no hidden layers is primarily learning to predict actions based on their frequency in the dataset, reflecting class imbalances rather than meaningful patterns and relationships between board states and actions. However, other than action 0 (the most common action), the values for the 3 other most common actions do not necessarily correspond to their frequency shown in the static prediction model. This indicates that while the control model is largely influenced by action frequency, it is also partially incorporating some relationships between the board state and actions, albeit in a seemingly limited and simplistic way.

Interestingly, a similar pattern is observed in RNN1's bias values, with the most common actions having positive bias values and less common actions having negative ones. However, the models differ in that RNN1 assigns the highest bias to action 6, with action 0 receiving the second-highest bias, whereas the baseline control model more closely mirrors the static prediction model with the highest bias assigned to action 0. The control model with an added hidden layer differs significantly from the baseline control model, RNN1, and the static prediction model. They are similar in that actions 2, 3, 4, and 7 all have negative biases, and actions 0,1,5 and 6 have positive values. Additionally, actions 0 and 6 are more similarly valued to the RNN than they are in the control. However, the control model with a hidden layer differs in that it positively values actions 8, 10, and 11, and values 3 as the least common action. This suggests that the baseline control model primarily predicts actions based on their frequency in the dataset, reflecting class imbalances rather than meaningful patterns or relationships between board states and actions. Overall, the control model with the added dense hidden layer exhibits less variation between actions and a smaller bias range compared to the baseline control model and RNN1.



As expected, RNN2 and RNN3 also significantly outperformed the control dense networks in terms of accuracy and loss. All three RNN models demonstrated better predictive performances as indicated by their lower validation losses (~1.05–1.06) compared to the control models (~1.22–1.24) and higher validation accuracies (~0.677–0.680) compared to the control models (~0.636–0.641). However, the RNN models showed only minimal differences in performance amongst each other despite the addition of LSTM layers. The plotted training loss and accuracy in Figure 10 indicate that RNN2 performed the best during training, followed by RNN1 and RNN3, a trend also shown in the plotted validation accuracies. The validation loss and accuracy values in the table (Figure 10) suggest a slightly different pattern where RNN1 achieved the lowest validation loss (1.049) and highest validation accuracy (0.680). RNN2 and RNN3 had slightly higher validation losses (1.052 and 1.063, respectively). In contrast, RNN2 and RNN3 exhibited slightly higher validation losses (1.052 and 1.063, respectively) and marginally lower validation accuracies (0.679 and 0.677). Interestingly, in both evaluations RNN3 performed the least effectively among the RNN models.

The constructed models were modified and re-trained using two distinct input configurations: previous action only and board state only. This was done to isolate the contributions of these individual components to the overall predictive performance which is shown in Figure 10 above and Figure 13 below.



These results demonstrate that all models achieve better performance, with lower losses and higher accuracies, when provided with the full game state. This indicates that incorporating the full game state provides the most comprehensive and informative input, allowing the models to better generalize and predict user actions. Moreover, when comparing the results of isolating previous action and board state, we can see that the previous action resulted in a better loss and accuracy across models.

We can begin our comparison by looking at the results of the two control models. The average accuracy for the two control models trained on gamestate is 0.6391 and the average loss is 1.232. Previous action for the controls has an average accuracy and loss of 0.5857 and 1.417 respectively, which are ~0.05 lower and ~0.185 higher than their corresponding gamestate values. Controls trained on board state alone resulted in an average accuracy and loss of 0.5747 and 1.422, which are ~0.06 lower and ~0.19 higher than their corresponding gamestate values. Though the difference is small, previous action seems to be a better predictor than board state for control models. Doing the same for our RNNs, the three models trained on game state have an average accuracy of 0.6786 and an average loss of 1.054. RNNs trained on previous action only have an average accuracy and loss of 0.668 and 1.085, these are ~0.01 lower and ~0.03 higher than their corresponding boardstate values. RNNs trained on board state only have an average accuracy and loss of 0.654 and 1.135, these are ~0.02 lower and ~0.08 higher than their corresponding boardstate values. As we can see previous action alone results in better accuracies and losses for our models than board state alone, indicating that it is more of a predictor and potentially contributes more to the performance of the gamestate models. However, the average accuracies and losses for our RNNs show smaller differences between input types than the control models. This suggests that their sequential processing capabilities allow them to extract more meaningful patterns from all features, reducing their reliance on specific input types like previous action or board state. However, the performance gap between the models trained on the full game state and those trained on either component individually highlights that both features contribute meaningful and complimentary information. The full game state allows the models to account for both the spatial relationships in the board state and the temporal dependencies from the previous action, enabling them to make more accurate predictions overall. Therefore, while the previous action is the stronger individual predictor, the combined use of both features is essential for maximizing the models’ predictive performance and fully capturing the complexity of the player’s decision-making process. As observed before, there are no significant differences between the performance of the three RNNs when using only previous action or board state. This further suggests that increasing the number of LSTM layers does not provide a significant advantage when the input data is limited to either the previous action or the board state alone.

Finally, to further assess how the RNNs were able to leverage the sequential data, and to what extent players' choices depend on their history, we looked at the average accuracy for each step in the sequence. After the control and RNN1 were trained, the average accuracy was calculated and plotted for each step in the sequence, from both the steps from the start and the steps from the end of the sequence as seen in Figure 14. Note that this was only done for RNN1 since RNN2 and RNN3 had no significant difference in performance.

Figure 14 above illustrates the accuracy at each step in the sequence, alongside a corresponding graph where the timesteps are plotted on a logarithmic scale. Contrary to our expectations, the two plots appear remarkably similar. In summary, the RNN starts at a higher accuracy (~0.47) than the control (~0.38), however, largely, the distribution of accuracy within the first ~200 steps was the same between the two models. Additionally, both models showed growing accuracy up to the first 200 positions. From positions 200 to 290, the accuracy of both models slightly decreases and seem to struggle to generalize until they both peak in accuracy around 300. Around position 350, several plateaus in accuracy appear for both models. Counting these as appearances of three or more steps clustered at the same accuracy, after position 350, the control has roughly 17 while the RNN has roughly 14. The RNN has its first value reach 1.0 at position 350, around 20 steps before the control which sees its first at 380. Interestingly, both models achieve their first 0 value at the same position of 580. After step 580, both models become entirely destabilized, where it either predicted correctly with a sharp drop to 0 accuracy, predicted incorrectly with spikes to 1, or plateaued at ~0.47, reflecting the accuracy of predicting the most common action.



In Figure 15 which plots the accuracy by reversed sequence position, aka steps from the end of the sequence, a similar, but inverted, trend emerges. Though the RNN has a slightly higher value at position 0 (the last step in a players sequence) difference in starting accuracy between the two models is marginal, especially when compared to that shown in Figure 14. Both start around 0.82. This value is significant since it is the same accuracy the models peaked at at step 300 in the plots above. The two models show a steady decrease in accuracy up until around position 200 where the player is 200 steps away from the end of the sequence. This mirrors the accuracy trends of the non-reversed sequence positions above, which show a growing accuracy as the position in the sequence approaches 200. Similarly, after position 200 the models begin to struggle to generalize. Plateaus in accuracy begin to appear at position 300 and are closer in count (roughly 19 for the control and 18 for the RNN) when compared than the plateaus discussed above and seen in Figure 14 which started at position 350. The RNN is the first to reach an accuracy of 1 at position 410 with the control following shortly after at position 430. The RNN is also the last of the two models to have an accuracy reach 0, with the control achieving this at position 460 and the RNN at 470. Finally, as it were above, both models completely destabilize at position 580 where they begin to only return accuracies of 0,1, or ~0.47.

To better visualize the difference in accuracies between the models, positions within the sequence were binned. Additionally, the average accuracy by both sequential and reversed sequence positions was calculated for the control network with an added hidden layer. Doing this ensured that these relationships were a result of the RNNs LSTM layers and their ability to pick up on sequential and temporal dependencies and not the addition of a hidden layer,



As seen in Figure 16, the RNN1 model outperforms both of the controls with higher accuracy at every position, both reversed and chronological. All models follow roughly the same accuracy trends across bins, in terms of where they increase, decrease, plateau, etc. Additionally, the control model with no hidden layer, though it had slightly higher accuracy than the baseline control, was nearly identical. This illustrates that the RNNs performance cannot simply be attributed to its hidden layer but rather the recurrent nature of its hidden layer. For the rest of this analysis the two controls performance will be referred to as one for a clearer comparison between the RNN and the controls.

On the plot on the left, accuracy across the first position (0) of user sequences starts at roughly the same point, ~0.44. The RNN improves much more quickly and steadily from position bin 0 to position bin 3-4 compared to the control which experiences a drop in accuracy at position bin 1-2 before increasing again. After this deviation, the two models decrease at the position 5-8 bin and then gradually increase up to position 17-32. However, both models plateau from position bins 17-32 to 257-512 with the RNN increasing slightly at bin 65-128 and 257-512 showing very faint variation while the control remains constant until the 257-512 bin where it slightly decreases. After position bin 257-512, both models show a dramatic increase in accuracy with the RNN exceeding 0.8 and the RNN reaching ~0.75.

When looking at the plot of reversed position bins, the average accuracy at the final step of each sequence is around 0.68 for the control and 0.74 for the RNN. From here, the accuracy dramatically increases at reversed positions 1-4 to 0.85 for the RNN and 0.83 for the control. This indicates that the model accuracy is generally highest when predicting the second, third, and fourth-to-last actions in a user's sequence but is notably less accurate when predicting the final action in a user's sequence. From reversed position bins 5-8 onwards both accuracies expectedly decline. Curiously, the reversed position bins show the same decline at action 5-8 that was seen in the non-reversed position bins. Additionally they begin to most dramatically decline at position 17-32 where the non-reversed plot plateaued.

Overall, these results highlight the RNN's superior ability to capture temporal and contextual patterns, especially when using the full game state as input. However, the findings also underscore limitations, such as both models’ struggle to predict the final action or generalize past certain positions.

# **Discussion**

This thesis demonstrated that Recurrent Neural Networks (RNNs), particularly those incorporating Long Short-Term Memory (LSTM) layers, outperform linear and nonlinear dense neural networks in predicting sequential player actions in *Hexxed.i*o gameplay. All models exhibited an accuracy which indicates they identified meaningful patterns in the data rather than relying on chance or the frequency of a single action. Notably, our RNN models were able to approximate human learning behavior with an average accuracy improvement of 6.19% over control models. Despite this improved ability to approximate human behavior, the RNNs failed to significantly replicate the abrupt, strategic cognitive shifts (leaps of insight) characteristic of human learning. Rather, the models relied heavily on patterns more explicitly present in the training data, failing to infer or generalize transferable strategies when handling longer, underrepresented sequences (>200 steps).

Based on the principles of imitation learning discussed in the literature review, we expected the models to generalize learned patterns to uncommon and more complex sequences. However, instead of accurately predicting users' strategic leaps for such sequences, the models demonstrated lower accuracy as a result of their reliance on explicitly trained patterns from shorter sequences. This limitation is also evident in the plotted biases observed across models (Figure 12), which closely resemble the results of the static prediction model (Figure 9). Additionally, sequence tracking plots (Figures 14 and 15) highlight how accuracy was heavily influenced by the distribution of sequences within the dataset.

##### Accuracy Trends and Data Bias

Though the similarities in accuracy trends across sequence lengths were initially surprising, they can be explained by data distribution. Accuracy at each timestep reflects the number of correct predictions over total valid predictions. Sequence tracking plots (Figures 14, 15, and 16) show a similar trend in accuracy between the RNN and baseline control model, particularly in the first ~120 steps. For the RNN, this incline in accuracy is aligned with our expectations, as temporal dependencies should become more apparent in later stages of player learning.

The baseline control model predicts actions based solely on the current game state, assigning weights to input features to maximize the likelihood of observed actions. Therefore, the similar trend in the control model, which lacks any ability to process sequential or temporal dependencies, was unexpected. We conclude that the similar trend in accuracies between the two models, especially within the first ~120 steps, can primarily be attributed to data distribution bias. Early steps are well-represented in the dataset, as seen in Figure 5, with the majority of sequences having under 150 steps. This overrepresentation of shorter sequences provided more training data, leading to better learning for both models. As such, the RNN’s high accuracy for early steps does not fully reflect its ability to handle complex dependencies but rather its ability to exploit these common patterns more successfully than the baseline controls.

Beyond 300 steps, accuracy destabilized, which was expected given the dataset’s 95th percentile for sequence length is 283, with only 80 sequences exceeding this. The observed plateaus starting around position 350 may correspond to regions where the model has seen enough data to make reasonable predictions but lacks variability to improve further. This is exactly where the RNNs should have outperformed the control model, as their ability to use temporal dependencies and long-term patterns should have given them an edge in handling these less common and more complex sequences. However, the fact that there wasn’t a noticeable performance gap suggests that while the RNNs are good at picking up on common patterns in shorter, well-represented sequences, they struggle to generalize to longer ones. This highlights a limitation in their ability to fully capture the deeper sequential relationships needed to truly stand out.

If the model had achieved greater success in predicting common actions at the beginning of the sequence due to their overrepresentation, the sequence tracking by reversed position would theoretically show a more pronounced difference in accuracy between the models. The RNNs, leveraging sequential dependencies, should have been better at making informed predictions compared to the control model; especially since in the final steps of the sequence, players are generally more informed about how the game works leading to more predictable patterns and decisions. This should reduce the variability in accuracy between the models, as the predictions rely on simpler, repetitive strategies. However, this was not the case. Both models displayed similar trends in accuracy across all positions, suggesting that the RNN's higher overall accuracy was more a result of exploiting common patterns rather than demonstrating a deep understanding of complex relationships. While the RNN utilized sequential dependencies to make more informed predictions of common actions, it failed to learn the more complex relationships —such as shifts in strategy—within those dependencies that are necessary to replicate the true leaps of insight characteristic of top-down human learning. This represents a key limitation in the LSTM layers ability to generalize and mimic human cognitive processes.

##### RNN advantages and key factors

Despite its limitations, the RNN consistently demonstrated higher accuracy and lower loss compared to the control across all stages of a sequence. This improvement is attributed to three key factors: temporal dependencies, integration of input features, and sequential problem solving.

Firstly, the majority of the RNN’s improved performance can be attributed to its ability to capture and retain long-term dependencies across player actions. As shown in Figure 11, temporal dependencies—rather than non-linearity—were the primary drivers of the RNNs’ superior ability to predict human actions, highlighting their importance in accurately modeling human-like decision-making. Introducing non-linearity to the control model demonstrated a slight improvement in performance, with a modest increase in accuracy (~0.005) and a decrease in validation loss (~0.02) compared to the baseline control model. This indicates that while a hidden layer can somewhat improve the model’s ability to capture more complex relationships between the board state and actions, it does not fundamentally address the limitations of the control model. In contrast, the RNNs significantly outperformed the control model, even when a hidden layer was added, reinforcing that temporal dependencies were the key factor driving their performance. This conclusion is further supported by the nonlinear control model's accuracy by binned sequence position, as shown in Figure 16, which reveals a trend nearly identical to the baseline control model. Notably, adding LSTM layers did not significantly enhance performance as hypothesized, with RNN1 achieving the highest validation accuracy (67.9%) across all models. This suggests that though additional layers may add capacity, they do not improve the model’s ability to capture patterns in the reduced input, likely a result of the simpler nature of the task. Interestingly, RNN3, which included three LSTM layers, consistently performed the worst among the RNN models. This suggests that increasing the number of LSTM layers beyond two may have even introduced unnecessary complexity, leading to slightly diminished generalization.

Secondly, the integration of input features demonstrably improved the RNNs ability to generalize better, highlighting the role of multimodal inputs in predictive accuracy of human behavior. As mentioned previously, all models performed best when provided the full game state (previous action and board state). The results of feature isolation (Figure 13) revealed that for all models, previous action is a more informative feature for predicting the next action compared to board state alone. In other words, the models are seemingly able to leverage the temporal dependency between consecutive actions more effectively than the spatial information provided by the board state. However, the performance gap between the models trained on the full game state (board state + previous action) and those trained on either component individually highlights that both features contribute meaningful and complimentary information. The full game state allows the models to account for both the spatial relationships in the board state and the temporal dependencies from the previous action, resulting in more accurate predictions overall. Interestingly, when training our models on isolated features, the RNN’s performance did not decline as sharply as controls, indicating RNNs can generalize better without being overly reliant on specific features. The similarity in performance across RNN1, RNN2, and RNN3 in these isolated scenarios further supports the conclusion that the complexity of our task, even when reduced to a single feature, can be adequately modeled with a single LSTM layer. Furthermore, Figure 16 shows that RNN1 consistently outperforms both control models with higher accuracy at every sequence position, whether reversed or chronological. This result further supports our conclusion that RNNs are better than baseline models at generalizing across timesteps and leveraging input features to improve predictions.

The third and final factor contributing to the RNN's performance, sequential problem solving, combines elements of the first two factors. Unlike the control models, which treat each action as an isolated event, the RNN’s ability to model sequential decision-making allows it to capture both temporal dependencies and contextual relationships between consecutive actions. This makes the RNN significantly more adept at mimicking human-like problem-solving behavior. In the context of level one of *Hexxed.io*, sequential problem solving is crucial because players must recognize and execute a specific sequence of actions to achieve the optimal outcome. For example, players need to learn that performing the action sequence [0,0,0,0,0,0,6] maximizes the reward for each subwave attempt. This sequence relies heavily on recognizing temporal patterns—understanding that six consecutive taps (0,0,0,0,0,0) sets up the necessary state for the swipe action (6) to complete the subwave. The RNN’s ability to retain and process this sequential structure across time steps gave it a clear advantage over control models, which lack any mechanisms to connect past actions to future predictions. For example, the RNNs could use their hidden states to "remember" that five 0s have already been taken, allowing them to then predict 6 more accurately. In contrast, the control models might recognize that action 0 often occurs after a certain board state but fail to predict that 6 is the logical conclusion after a sequence of five 0s. While all players in the dataset demonstrated learning by successfully passing level one, this does not necessarily provide sufficient evidence that the RNNs captured the broader temporal dependencies contributing to leaps of insight. Instead, the models may have focused on the specific dependencies of the [0,0,0,0,0,6] sequence and its direct role in winning, rather than emulating the deeper, more generalized learning processes characteristic of human insight.

##### Limitations and Future Directions

These results reflect a broader challenge AI faces: difficulty generalizing from limited or unevenly distributed data compared to humans, who rely on abstract and transferable strategies. The extent to which the model successfully captured the temporal dependencies that underpin leaps of insight remains somewhat uncertain, mostly because of the small size of our dataset. This data constraint represents a significant limitation, and future studies should aim to address it by training on action sequences from a larger pool of players with more diverse ranges of action sequences. A larger dataset could help mitigate the effects of data distribution bias, allowing for a more thorough analysis of the RNNs generalizability.

Another key limitation of this study is that the RNNs lacked access to a crucial contextual factor that human players had: score values. Score values likely played an important role in informing Hexxed players’ decisions and contextual understanding of the task. Score feedback likely served as a critical signal for players to evaluate the success of their current strategies, guiding them toward leaps of insight. Without this feedback mechanism, the RNNs were unable to replicate the reward-driven dynamics present within Hexxed player learning and limited their ability to model the cognitive processes behind leaps of insight in this context. Future research should explore integrating score feedback into RNN training to more effectively capture these reward-driven dynamics. Future studies should explore hybrid models that integrate reinforcement learning with RNNs, combining bottom-up pattern recognition with top-down goal-driven learning to make AI systems more human-like in how they learn and adapt. In Linda L. Emberson’s chapter "*How does experience shape early development? Considering the role of top-down mechanisms*," she speaks about the dynamic interaction between perception and cognition. The chapter explores how perception is not a static process but dynamically shaped by the brain’s current goals, motivations, and stored knowledge. AI models like LSTMs rely on sequential dependencies and backpropagation to "learn" patterns, but they lack intrinsic top-down processes like goals or motivations. Introducing this through a hybrid RNN-RL model could integrate reinforcement learning’s reward-based feedback could provide a form of goal-directed optimization, helping the model evaluate past strategies and shift behavior dynamically, much like how humans use performance feedback to refine their understanding and develop new insights. This approach could help AI move beyond simply memorizing patterns, allowing it to create strategies that are more purposeful and goal-oriented—bringing it closer to replicating leaps of insight and top-down processing.

# **Conclusion**

This study demonstrated the ability of Recurrent Neural Networks to approximate aspects of human learning behavior through sequential decision-making tasks in the game *Hexxed.io*. By leveraging temporal dependencies, RNNs, particularly the single-layer LSTM model (RNN1), outperformed baseline control models in predicting player actions. The potential of RNNs and imitation learning frameworks to capture these dynamics is clear; however, a larger and more diverse dataset was ultimately needed to fully explore their potential for modeling the strategic shifts and top-down characteristics of human learning during Hexxed gameplay. RNNs offer an important step toward introducing top-down learning mechanisms into AI systems, however, their limitations make it clear that we cannot isolate a single facet of human learning, such as sequential processing, and expect it to sufficiently capture the interplay of adaptability, exploration, and context-aware behavior which define leaps of insight. To address these limitations, future research should focus on integrating hybrid models that combine the RNNs with other bottom-up mechanisms that address different but complementary aspects of learning, such as RL or reward-modulated neural networks. Such an approach could allow AI to take advantage of sequential dependencies while also introducing goal-oriented behaviors that refine strategies based on the rewards and outcomes of exploration, much like humans do. This potential for combining bottom-up mechanisms to emulate top-down dynamics in human learning ultimately points to exciting opportunities for creating AI that can better capture the complexity that defines human intelligence.

# **Acknowledgments**

I would like to thank my Professor Dr. Gautam Agarwal for their guidance and support throughout this project. I would also like to thank Professor Gabriel Hope and for his support with the model design and implementation. I would like to recognize Tiago Quandera, Guillermo Mendoza-Franco, and my other Beatlab teammates who helped contribute to Hexxed and this project. Finally, I thank my mother for convincing me to take my first computer science class, without which this project would not have been possible.

# **Bibliography**

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140–1144. https://doi.org/10.1126/science.aar6404.

Ullman TD, Goodman ND, Tenenbaum JB. Theory learning as stochastic search in the language of thought. Cognitive Development. 2012;27(4):455-480. doi:10.1016/j.cogdev.2012.07.005

Quendera, T., Deng, D., Hamidi, M., Bergomi, M., Mainen, Z. F., & Agarwal, G. (2022). Humans learning a complex task are picky and sticky. *Proceedings of the Cognitive Science Society Conference*.

Sun, Ron, and Xi Zhang. "Top-Down versus Bottom-Up Learning in Skill Acquisition." *Cognitive Systems Research* 5, no. 1 (2004): 63–89. <https://doi.org/10.1016/j.cogsys.2003.07.001>.

James Fodor. (2022, June). “Neuroscientist discusses differences in AI and human intelligence.” *Tech Xplore*. <https://techxplore.com/news/2022-06-neuroscientist-differences-ai-human.html>

Slagter, H. A. (2023). Perceptual learning in humans: An active, top-down-guided process. Behavioral and Brain Sciences, 46, e406. doi:10.1017/S0140525X23001644

Ahammed, G. (2024, March 28). Understanding imitation learning: How AI agents mimic behavior. *Nothing But AI*. <https://nothingbutai.com/understanding-imitation-learning-how-ai-agents-mimic-behavior/>

Geisler, B. (2004). Integrated machine learning for behavior modeling in video games. In Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press, Menlo Park (pp. 54-62).

Zewe, A. (2024, April 19). Building a better AI helper starts with modeling irrational behavior. MIT News. Retrieved December 6, 2024, from https://news.mit.edu/2024/building-better-ai-helper-starts-with-modeling-irrational-behavior-0419

Gibson, E. J. (1988). Exploratory behavior in the development of perceiving, acting, and the acquiring of knowledge. Annual Review of Psychology, 39, 1–41. https://doi.org/10.1146/annurev.ps.39.020188.000245

Liu, Y. (H.). (2020, December 15). Understanding the mechanism and types of recurring neural networks. *Open Data Science*. <https://opendatascience.com/understanding-the-mechanism-and-types-of-recurring-neural-networks/>

Graves, A. (2012). Supervised sequence labelling with recurrent neural networks. Springer. https://doi.org/10.1007/978-3-642-24797-2

Ubal, C., Di-Giorgi, G., Contreras-Reyes, J. E., & Salas, R. (2023). Predicting the Long-Term Dependencies in Time Series Using Recurrent Artificial Neural Networks. *Machine Learning and Knowledge Extraction*, *5*(4), 1340-1358. <https://doi.org/10.3390/make5040068>

Sharma, A. K. (2023, February 15). Predicting human behaviour activity using deep learning LSTM. Medium. Retrieved from https://medium.com/@chataks93/predicting-human-behaviour-activity-using-deep-learning-lstm-fff9030b82e7

Tang, S., Samuel, S., & Li, Z. (2023). Detecting Atypical Test-Taking Behavior with Behavior Prediction Using LSTM. Psychological Test and Assessment Modeling, 65(1), 76-124.

Conroy, D., Wyeth, P., & Johnson, D.M. (2011). Modeling player-like behavior for game AI design. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. https://doi.org/10.1145/2071423.2071434

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30. https://arxiv.org/abs/1706.03762

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19), 2623–2631. https://doi.org/10.1145/3292500.3330701

Chatakonda, S. (2023, February 15). *Predicting human behaviour activity using deep learning LSTM*. Medium. Retrieved December 6, 2024, from <https://medium.com/@chataks93/predicting-human-behaviour-activity-using-deep-learning-lstm-fff9030b82e7>

Chollet, F., & contributors. (n.d.). *Training and evaluation with the built-in methods*. Keras. Retrieved December 2, 2024, from <https://keras.io/guides/training_with_built_in_methods/>

Olah, C. (2015, August). Understanding LSTMs. *Colah’s Blog*. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Emberson L. L. (2017). How Does Experience Shape Early Development? Considering the Role of Top-Down Mechanisms. Advances in child development and behavior, 52, 1–41. https://doi.org/10.1016/bs.acdb.2016.10.001

Emberson L. L. (2017). How Does Experience Shape Early Development? Considering the Role of Top-Down Mechanisms. *Advances in child development and behavior*, *52*, 1–41. https://doi.org/10.1016/bs.acdb.2016.10.001