Optimizing Strategies Against Uncertainty: A Comparative Study of Advanced Learning Algorithms in Stochastic Environments with Visibility Constraints

AUTHOR: MEGHA VISWANATH

UNDER THE GUIDANCE OF DR. SHAUNAK D BOPARDIKAR

*Abstract*— In the domain of artificial intelligence, the game "Catch" serves as a crucial testbed for refining learning algorithms. This study investigates the performance of Q-Learning, Fictitious Play, Deep Q-Networks (DQN), and Neural Networks within this environment, particularly under the additional complexity of visibility constraints and different types of falling objects. Q-Learning and DQN address complexities where the movement of the falling ball can change direction (left or right) with set probabilities, enhancing the agents' adaptive strategies. The Neural Networks model, with visibility constraints, evaluates strategic decision-making when different types of balls fall unpredictably. The research demonstrates the scalability and adaptability of these models, emphasizing DQN's capability to manage the expanding state space and improve strategic decision-making in stochastic settings.

Keywords— Q-Learning, Deep Q-Networks, Fictitious Play, Neural Networks, Stochastic Environments, Visibility Constraints, Learning Algorithms, Adaptive Strategies, Probabilistic Movement.

# Introduction

The study of reinforcement learning is a cornerstone of artificial intelligence, providing insights into how algorithms can adapt and optimize behavior in dynamic environments. A commonly used testbed for exploring these algorithms is the game "Catch," where the simplicity of the game mechanics contrasts sharply with the complexities introduced by advanced learning methodologies. This research began with the basic application of Q-Learning, a foundational algorithm in reinforcement learning, which is typically the starting point for those new to the field.

* *Evolution of Complexity in Learning Environments*

Initially, the application of Q-Learning in the game of "Catch" served to familiarize the foundational concepts of reinforcement learning—primarily how an AI agent learns to maximize its score by catching falling objects, which move according to a set of probabilistic rules. As the research progressed, increasing levels of complexity were introduced to test the limits of Q-Learning. This included enhancing the unpredictability of object movements and expanding the state space, which rapidly highlighted the limitations of Q-Learning in handling larger, more complex environments.

* *Transition to Deep Q-Networks (DQN)*

To address the escalating complexity and the exponential growth of the state space, the study advanced to the use of Deep Q-Networks (DQN). DQNs leverage neural networks to approximate the Q-value functions, which significantly mitigates the challenges posed by larger state spaces. However, this advancement also introduced greater computational demands, reflecting a trade-off between complexity management and processing requirements.

* *Exploring Visibility Constraints and Environmental Dynamics*

The research then ventured into what might happen in hypothetically immense environments. It posited a scenario where the agent's vision is limited to a 5x5 section within a larger 15x10 environment. This setup challenged the agent to not only manage its immediate space but also to make strategic decisions about when to move to adjacent sections based on the potential rewards. The introduction of two types of balls—red and blue, with different point values—added a layer of strategic depth. This element was designed to test the agent's ability to prioritize actions based on the perceived value of objects within its constrained view.

* *Research Questions and Objectives*

This progression naturally led to the formulation of several research questions:

1. How does an AI agent adapt its strategy in response to increasing complexities in the game environment?
2. Can DQNs manage the expanded state space more effectively than traditional Q-Learning in terms of computational efficiency and learning performance?
3. What strategies does an AI agent employ when faced with visibility constraints and variable reward systems?
4. How do the performances of AI agents compare to human players in similar game settings?

* *Contributions*

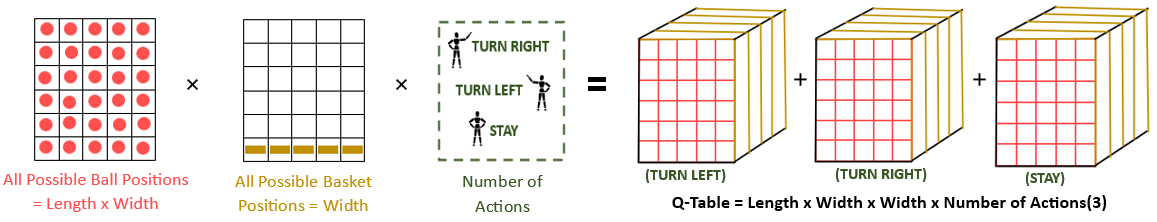
This report aims to contribute a structured exploration of these questions, providing a comprehensive analysis of the behavior and performance of AI agents as they progress from simple to highly complex game scenarios. By systematically increasing the complexity and observing the corresponding adaptations in AI strategies, this study not only enriches our understanding of reinforcement learning dynamics but also tests the boundaries of algorithmic adaptability in artificial intelligence.

# Methods

## **Q-Learning Algorithm**

* *Overview*

Q-Learning is a type of model-free reinforcement learning algorithm that enables an agent to learn the value of an action in a particular state within a stochastic environment. It does



***Fig 1. State Space Expansion***

not require a model of the environment and works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. This method is particularly favoured in problems where the model dynamics are unknown and the state and action spaces are discrete.

* *How Q-Learning Works*

Q-Learning operates by updating the Q-values (action-value pairs) that are stored in a Q-table. The Q-table is a matrix where each row represents a possible state, and each column represents a possible action. The agent uses this table as a reference to select the best action based on the Q-values associated with the current state.

The core of the Q-Learning algorithm involves updating the Q-values stored in the table using the equation:

Where:

* is the current state,
* is the action performed,
* is the reward received after performing the action,
* is the learning rate,
* is the discount factor that weighs the importance of future rewards,
* is the highest Q-value for the next state .

This process is repeated for each action taken in the environment, gradually improving the policy until the Q-values converge to their optimal values.

* *Q-Table and Q-Function*

The Q-table is crucial in storing and updating the Q-values based on the agent's experiences in the environment. It allows the agent to start with initial guesses for the Q-values and update them based on the feedback (reward signals) received from the environment.

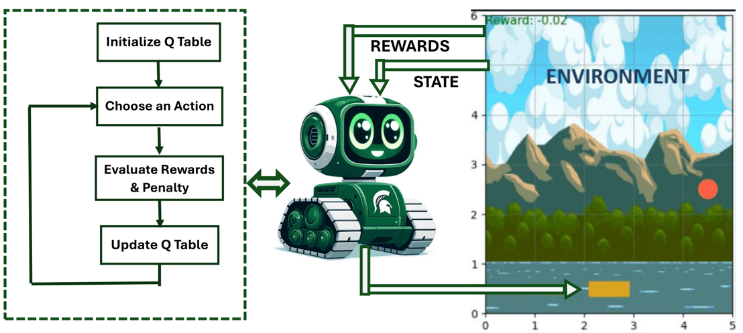
The Q-function represents the expected future rewards that can be obtained from taking an action in a given state, following a certain policy. It is a way of estimating the reward a player can expect to accumulate over the future, starting from that state.

* *Implementation in "Catch" Game*

In the "Catch" game:

* The state is defined by the positions of the ball and the basket.
* Actions are moving the basket left, right, or staying in place.
* The Q-table is initialized with all zeros and updated continuously as the game progresses and the agent learns from its actions.

The figures provided illustrate the dimensions of the Q-table which incorporates all possible positions of the ball and basket, along with the different actions. These visualizations help in understanding how the Q-table expands with the complexity of the state and action space in the game.



***Fig 2. Q Algorithm***

As the state space increases, Q-Learning becomes computationally expensive, which led to the exploration of DQNs to handle larger state spaces more effectively.

The next section will delve into the intricacies of the Deep Q-Network model and how it addresses the challenges faced by traditional Q-Learning methods.

## **Fictitious Play for Model-Free Settings**

* *Background and Implementation from Existing Work*

The implementation of fictitious play in this project draws directly from the approach detailed by Sayin, Parise, and Ozdaglar in their seminal paper, "Fictitious Play in Zero-Sum Stochastic Games." This paper presents a novel integration of fictitious play with Q-learning, tailored for model-free environments in stochastic games. Fictitious play, traditionally used in strategic-form games, involves players repeatedly choosing strategies that are best responses to the empirical distribution of the opponent's past actions. This method is adapted to accommodate the complexities of dynamic environments where actions influence not only immediate rewards but also future states and ongoing payoffs.

* *Overview*

In zero-sum stochastic games, the adaptation involves

players forming beliefs about the opponent's strategy and their own potential future rewards, captured by the Q-function. These beliefs are updated based on observed actions and outcomes. A key feature of the adaptation is the dual timescale update process: strategies are updated more frequently than Q-functions, allowing for a responsive adjustment to strategies while maintaining a stable progression towards optimal long-term payoffs.

* *Key Components*

1. Strategy Update:

Players update their strategy at state using a learning rate , adjusting towards the best response based on the empirical distribution of the opponent’s actions, .

1. Q-Function Update:

The Q-function for a given state and action is updated to reflect the immediate reward plus the discounted value of the best future payoff, based on the transition probabilities and the maximum payoff achievable from subsequent states.

* *Application in the "Catch" Game*

In the "Catch" game, the fictitious play algorithm is implemented with two agents: one trying to catch the ball and the other throwing the ball. Both agents adjust their strategies dynamically based on the observed actions and patterns of the other agent.

* Catching Agent:

A cartoon of robots with speech bubbles

Description automatically generatedThe catching agent adjusts its strategy dynamically based on the observed actions and patterns of the throwing agent. This involves forming beliefs about the best positions and movements to catch the ball, based on the empirical distribution of where the throwing agent aims the ball.

***Fig 3. Fictitious Play Depiction***

* Throwing Agent:

The throwing agent decides which column to throw the ball into, adjusting its strategy based on the catching agent's movements. The throwing agent forms beliefs about the catching agent's positioning and updates its strategy to make it more challenging for the catching agent to predict and catch the ball.

This learning dynamic enables both agents to optimize their strategies over time. The catching agent aims to improve its positioning and decision-making to maximize successful catches, while the throwing agent adjusts its strategy to minimize the chances of being caught. By detailing the theoretical underpinnings and practical application, this report underscores the utility and effectiveness of this method in enhancing AI's strategic learning capabilities in complex scenarios. The application of fictitious play in the "Catch" game demonstrates the algorithm's robustness in handling dynamic and adversarial environments.

* *Future Improvements*

For future improvements, the throwing agent could also vary the trajectories and speeds of the throws to further challenge the catching agent. Additionally, implementing probabilistic movements of the balls (e.g., balls that can change direction mid-fall with certain probabilities) would increase the game's complexity. These enhancements would provide a more robust test of the fictitious play algorithm's effectiveness in even more dynamic and adversarial environments.

## **Deep Q Networks (DQN)**

* *Overview*

Deep Q-Networks (DQN) represent a significant advancement in reinforcement learning, integrating deep neural networks with Q-Learning principles. Introduced to manage the challenges of large state spaces where traditional Q-Learning would struggle, DQNs use deep neural networks to approximate the Q-value function, enabling the handling of high-dimensional sensory inputs directly.

* *How DQNs Work*

DQNs extend the Q-Learning algorithm by replacing the Q-table with a neural network that estimates Q-values. This approach allows the agent to generalize and infer Q-values for unseen states, effectively addressing the scalability issues associated with large state spaces.

The primary components of DQN are:

1. Q-Function Approximation:

Here, 𝜃 represents the parameters of the neural network, which are optimized to approximate the optimal Q-value function .

1. Experience Replay:

The agent stores its experiences in a replay buffer and samples mini-batches of experiences to update the network. This helps break the correlations between consecutive experiences and leads to more stable learning.

1. Target Network:

To further stabilize training, DQN uses a target network with parameters , which are periodically updated to the values of the main network's parameters . This helps in reducing oscillations and divergence during training.

The Q-network is trained to minimize the loss:

where is the replay buffer, 𝛾 is the discount factor, and 𝑟 is the reward.

* *Implementation in the “Catch” Game*

In the "Catch" game, the DQN algorithm is used to control the catching agent. The state space includes the position of the catching agent and the falling ball, and the action space consists of moving left, right, or staying in place. The DQN agent learns to maximize its score by catching the falling objects while managing the large state space effectively through neural network approximations.

* **State Representation:** The state 𝑠 is represented as a combination of the positions of the catching agent and the falling ball. The state is a 2D grid, where the agent and the ball are positioned.
* **Action Space:** The actions 𝑎 available to the agent are moving left, moving right, or staying in place.
* **Reward Structure:** The agent receives a positive reward for catching the ball and a negative reward for missing it.
* **Neural Network Architecture:** The neural network used to approximate the Q-value function consists of a Flatten layer to convert the 2D grid state into a 1D vector, followed by two Dense layers with 24 neurons each and ReLU activation. The output layer has neurons equal to the number of actions, with a linear activation function. This architecture allows the agent to effectively generalize and infer Q-values for unseen states, addressing the scalability issues associated with large state spaces.

The DQN algorithm involves the following key steps:

* **Initialization:** The main and target Q-networks are initialized with random weights. The replay buffer is also initialized.
* **Episode Loop:** For each episode, the environment is reset to its initial state. The agent interacts with the environment over a sequence of time steps, choosing actions based on an epsilon-greedy policy to balance exploration and exploitation.
* **Action Selection:** At each time step, the agent selects an action based on the current state using the epsilon-greedy policy. This policy involves selecting a random action with probability 𝜖*ϵ* and the action with the highest predicted Q-value with probability 1−𝜖.
* **Experience Storage:** The agent takes the selected action, observes the reward and the next state, and stores the experience (𝑠,𝑎,𝑟,𝑠′) in the replay buffer.
* **Mini-batch Training:** At each training step, a mini-batch of experiences is sampled from the replay buffer. The agent calculates the target Q-value for each experience using the target network and updates the main Q-network by minimizing the loss between the predicted Q-values and the target Q-values.
* A diagram of a diagram of a data processing process

  Description automatically generated with medium confidence**Target Network Update:** The weights of the target network are periodically updated with the weights of the main network to ensure stability in learning.

***Fig 4. DQN Implementation***

This structured approach enables the DQN agent to effectively learn and adapt its strategy to maximize the score in the "Catch" game, handling the complexities of the large state space and dynamic environment. The use of neural networks for Q-value approximation, combined with experience replay and target network updates, allows the agent to achieve robust performance in catching the falling objects.

## **Neural Network Strategies in Partially Observable Game Fields**

This method utilizes two neural networks to enhance the strategic decision-making of an agent operating within a partially observable game field. The agent, constrained by a restricted view of the environment, relies on the Point Assessment Network (PAN) and the Section Transition Network (STN) for decision-making.

A grid with blue and red dots

Description automatically generated

70 points if I stay. Should I move?

***Fig 5. Partially Observable Sections***

* *Neural Networks*
* **Point Assessment Network (PAN)**

Objective: Assess potential points from the current restricted view.

Architecture: The PAN is a feedforward neural network with an input layer corresponding to the flattened representation of A bar code with green lines

Description automatically generatedthe restricted view, one hidden layer, and an output layer that predicts the potential points.

Training: The network is trained on simulated data where each input is a restricted view and the output is the potential points for that view. This supervised learning approach allows the PAN to accurately estimate the value of visible objects.

* **Section Transition Network (STN)**

Objective: Decide whether the agent should move to another section based on the points assessed by PAN and other factors.

Architecture: The STN is a feedforward neural network that takes as input the points assessed by PAN along with the restricted view. It consists of an input layer, one hidden layer, and an output layer providing probabilities for three possible actions: staying in the current section, moving left, or moving right.

Training: The STN is trained on data that includes the restricted view, the points assessed by PAN, and the actions taken by the agent. The network learns to predict the optimal action based on these inputs, allowing the agent to make informed decisions about movement.

* *Integration*

At each step, the agent uses PAN to evaluate the potential points from its current restricted view. This information, combined with the current view, is fed into the STN, which decides the agent's next action (stay, move left, or move right). This dual-network approach ensures the agent can navigate the partially observable environment effectively, maximizing its total score through strategic point assessment and movement decisions.

# RESULTS

* *Q-Learning*

The Q-Learning algorithm was tested in a 5x6 game environment, and its performance was evaluated over 8,000 episodes. Initially, the agent experienced many misses as it explored different actions and learned the optimal strategy. This is depicted in Fig. 6, which shows the learning progress of Q-Learning over the initial 150 episodes. The frequent transitions between catch (1) and loss (0) indicate the exploration phase.

A screenshot of a graph

Description automatically generatedFig. 7 demonstrates the cumulative learning progress of Q-Learning over 5000 episodes. The cumulative catches increase steadily, indicating that the agent's performance improves as it continues to learn from its experiences. This graph highlights the necessity for extensive training (over 8,000

***Fig 8. Performance of Q-Learning with Varying Probabilities***

***Fig 6. Learning Progress of Q-Learning***

A graph with a green line

Description automatically generated

***Fig 7. Cumulative Learning Progress of Q-Learning***

episodes) for the agent to perform well, especially as the environment size increases.

The performance of this algorithm was also evaluated with varying probabilities of the ball moving straight downward (with the rest being split between left and right sideway movements) as shown in Fig. 8. The agent performs well when objects fall mostly downward (probability close to 1.0) or purely sideways (probability close to 0.0). A mix of movements (probability around 0.5) leads to more losses, indicating the agent's adaptability to different patterns of unpredictability.

Q-Learning is effective for environments with smaller state spaces and predictable dynamics. However, its performance is challenged by larger state spaces and increased stochasticity in the environment. This highlights the need for more advanced methods like Deep Q-Networks (DQN) to handle such complexities.

* *Fictitious Play*

In the Fictitious Play model, we analyzed the dynamic interaction between two agents: Player 1 (the catching agent)

and Player 2 (the throwing agent). The goal was to observe how each player adapts their strategy over time and how these strategies converge towards a Nash equilibrium.

A graph showing a line graph

Description automatically generated with medium confidence

***Fig 9. Difference in Continuation Payoffs Over Episodes***

Figure 1 shows the difference in continuation payoffs between Player 1 and Player 2 over the course of the episodes.

Key Observations:

* Initial Variability: The early episodes exhibit significant variability in the difference in continuation payoffs, representing the exploration period where both agents adjust their strategies based on the observed actions of their opponent.
* Convergence to Equilibrium: As episodes progress, the difference stabilizes around zero, indicating that both players' strategies converge towards a Nash equilibrium.

The results highlight the dynamic and adaptive nature of the strategies employed by both agents. The convergence towards a Nash equilibrium demonstrates the effectiveness of the Fictitious Play algorithm in enabling agents to learn and adapt their strategies in a competitive, model-free environment.

* *Deep Q Networks*

The Deep Q-Networks (DQN) model was implemented to manage the challenges of large state spaces by integrating deep neural networks with Q-Learning principles. This model was applied to the "Catch" game, where the objective was for the agent to catch falling objects in a dynamic and stochastic environment.

A graph showing a line graph

Description automatically generated with medium confidence

***Fig 10. Moving Average of the agent’s rewards over time***

Rapid Learning and Adaptation: Unlike the Q-Learning algorithm, which required training over 10,000 episodes to achieve good performance, the DQN model shows significant improvement within just 100 epochs. This demonstrates the efficiency of the DQN model in learning and adapting to the game environment quickly.

Consistent Improvement: The moving average plot indicates a steady increase in the average reward, highlighting the agent's ability to learn and refine its strategy effectively. The average reward stabilizes around higher values, showing the agent's successful convergence to an optimal strategy.

Robust Performance: The green shaded area represents the variability in rewards, which decreases over time, indicating that the DQN agent not only learns quickly but also achieves stable performance.

The DQN model effectively handles large state spaces and stochastic environments, showing rapid learning and stable performance within 100 epochs. However, running this algorithm requires substantial computational resources, and leveraging High-Performance Computing Clusters (HPCC) is essential. This underscores the robustness and scalability of DQN, making it a superior choice for tasks requiring quick adaptation and learning.

* *Neural Network Strategies in Partially Observable Game Fields*

The Neural Network Strategies model employs two neural networks to enhance the strategic decision-making of an agent in a partially observable game field. The agent uses a Point Assessment Network (PAN) to evaluate potential points from its restricted view and a Section Transition Network (STN) to decide whether to move to another section based on the points assessed and other factors.

A graph with red dots

Description automatically generated

***Fig 10. Impact of Strategic Section Changes on Rewards***

The plot indicates that the AI agent learns to minimize section changes, making such decisions primarily when the rewards dip below 5 points. This behavior suggests that the agent becomes more strategic over time, preferring to stay in the current section unless a significant reward decrease prompts a move.

The red dots in the plot represent the episodes where the agent decides to change sections. The agent's decisions are more frequent in the earlier episodes and become less frequent as it learns, highlighting the increased efficiency in its decision-making process.

Over time, the rewards stabilize around higher values, indicating that the agent's strategy is effective in maximizing its rewards while minimizing unnecessary section changes.

This model shows how neural networks enhance strategic decision-making in complex, partially observable environments. The AI agent strategically performs well by sticking to consistent rewards and only exploring new strategies if performance dips or if it persistently achieves higher rewards, typically exploring randomly every 30 episodes or so, depending on the exploration rate set. This contrasts with human tendencies to continually seek better rewards.

# ACKNOWLEDGMENT

I would like to thank Professor Bopardikar for the opportunity to learn and explore these algorithms, for introducing me to and guiding me in understanding these concepts from a game theory perspective, and for helping me explore my theories. I also extend my gratitude to the professors and mentors of MSU's CSE, ECE, and CMSE departments for their support and guidance in helping me understand the supporting concepts and clearing my doubts.

# REFERENCES

[1] M. O. Sayin, F. Parise, and A. Ozdaglar, “Fictitious Play in Zero-Sum Stochastic Games,” arXiv preprint arXiv:2010.04223, 2020.

[2] D. Holowaychuk, "Q-Learning Algorithm — From Explanation to Implementation," Towards Data Science, Oct. 9, 2018. [Online]. Available: https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187.