Evaluation of deep reinforcement learning and its application through a case study in computer games

by

Dhairya Gogri

In partial fulfilment of the requirements

for the degree of

MSc in

Software Development



Department of

Computer and Information Sciences

August, 2023

Abstract

This report explores the application of Q-learning, a model-free reinforcement learning technique, to the iterated prisoner's dilemma (IPD). The IPD provides an interesting testbed for evaluating whether independent learning agents can develop optimal conditional cooperative strategies solely through trial-and-error experience. The report reviews key literature on Q-learning for repeated games, generalization techniques, deep reinforcement learning, and multi-agent learning in social dilemmas. A systematic investigation is undertaken applying Q-learning with various enhancements like neural function approximation, experience replay, and hyper parameter tuning to the IPD within the Axelrod library framework. Multiple learning agents are trained against a diverse pool of opponents and then compete against each other in tournaments. Rigorous statistical analysis quantifies performance gains from the techniques, and learning curves provide insights into training progression. The experiments demonstrate that even basic Q-learning can achieve effective cooperation against certain strategies, but inconsistencies persist against punishing and stochastic opponents. More sophisticated deep RL methods are needed to fully master complex multi-agent strategic scenarios. The report provides a template for pragmatically improving reinforcement learning performance through incremental innovation. It highlights important considerations like statistical rigor, hyper parameter tuning, and quantitative evaluation to tackle problems of comparable complexity.

**Declaration**

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc in Software Development of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself.

Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

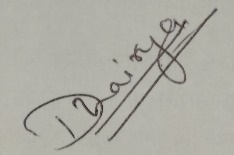
I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.

Yes [ X ] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is 13808.

I confirm that I wish this to be assessed as a Type 1 2 3 4 5 Dissertation.

Signature:

Date: 13-August-2023

Acknowledgements

I would like to express my deepest gratitude to my project supervisor, Dr. Jules Hedges, for his invaluable guidance, insights, and support throughout this research project. His extensive expertise in artificial intelligence and machine learning has been instrumental in shaping my understanding of key concepts like reinforcement learning and deep neural networks. I sincerely appreciate the time he dedicated to providing thoughtful feedback that challenged me to strengthen the technical rigor and evaluation of my experiments.

I would also like to thank the head of the Department of Computer and Information Sciences, Professor Clemens Cupke, for enabling access to the computational resources and tools needed to effectively undertake this project. The department's computing facilities were crucial for efficiently executing the many tournaments required for proper assessment.

Finally, I extend my thanks to the open source Axelrod library community. The capabilities offered by this robust framework immensely facilitated the implementation and reproducible analysis of Q-learning on the iterated prisoner's dilemma. The availability of thoroughly tested opponent strategies allowed comprehensive evaluation of the learning agent's performance.

The guidance, tools, and knowledge shared by my supervisor, department, and the open source community have been invaluable for completing this research project. I am deeply grateful for their support.

Contents

**1Introduction and Rationale……………………………………………………….……..7**

**1.1Investigating Q-learning in the IPD is important for several reasons…………………………………….…………………………………….…8**

**1.2 Implementing Q-learning to the IPD presents relevant as well as compelling research area for various reasons….………………………..…8**

**1.3 Goals and Purposes…………………………………………………………9**

**2. Literature review……………………………………………………………………..….9**

**2.1 Q-Learning for Repeated Matrix Games…………...............….…….…10**

**2.2 Generalization and Exploration…………………………………………..10**

**2.3 Memory and Communication……………………………………………..11**

**2.4 Deep Reinforcement Learning……………………………………………11**

**2.5 Evolution of Cooperation………………………………………………….12**

**2.6 Multi-Agent Reinforcement Learning……………………………………12**

**2.7 Conclusion…………………………………………………………………...13**

**3. Applying Q-learning to IPD…………………………………………………………...15**

**3.1 Previous Studies……………....……………………………………………15**

**3.2 Gaps in Current Research………………………………………………...15**

**4.Methodology………………………………………………………………..….16 4.1 Axelrod library architecture…………..…………………………………..17**

**4.2 Creating the Q-Learner class and Initializing Parameters…………..18**

**4.3 Brief on the Q-learning algorithm implementation…………………...19**

**4.4 Q-learning Strategy Implementation…………………………………….21**

**4.5 Updating Match Attributes and Storing Metrics………………………22**

**4.6 Plotting and Statistical Analysis Functions……………………………23**

**4.7 q\_learning\_tournament()……………………………………………….....25**

**4.8 Create an empty list to store mean scores across iterations………26**

**4.9 Calculation and Storing of Metrics………………………………………27**

**4.9.1 T-Test and Learning Curve Plotting…………………………………...27**

**4.9.2 Learning Curve Plotting (After All Iterations)………………………..27**

**4.9.3 Importance of p-value…………………………………………………....28**

**4.9.4 Importance of t-statistic…………………………………………………28**

**4.9.5 Other statistical evaluations………………………………………...….29**

**4.10 Analyzing the scores and rankings of your Q-learning agent**

**and the opponents……………………………………………..............30**

**5. Implementation 28**

**5.1 Implementing Q learning against Cooperator, Defector, Tit-for-tat, Grudger, Random strategies………………………………………………….30**

**5.2 Experimentation with the hyper parameters with linear**

**epsilon decay…………………………………………………………………….32**

**5.3 Using function approximation with ReLU activation function……..33**

**5.4 Q table printed with more reps and turns………………………………35**

**5.5 Adding a replay buffer as storage……………………………………….37**

**5.6 Using a separate target network to stabilize learning feedforward neural network with 2 hidden layers,2-element history vector…………39**

**5.7 improve training, evaluating every 10 episodes during training Tracking average rewards from evaluation episodes Plotting rewards**

**to see training progress. Different parameters…………………………...40**

**5.8 Calculating metrics over 5 iterations on the vanilla Q-learner……..42**

**5.9 Calculating metrics over 50 iterations to observe the changes……44**

**5.10 Q learner against various other strategies…………………………...46**

**5.10.1 Q-learner compared to Alternator, Anti Tit-for-tat, Bully, CooperatorHunter, CyclerCCCD…...…………………………………………46**

**5.10.2 Q-learner against Handshake, Inverse, Golden, Pi,WinStayLoseShift opponents……………………………………………...48**

**5.10.3 Q-learner against FoolMeOnce, OnceBitten, Willing, Hopeless, Desperate opponents…………………………………………………………..50**

**5.10.4 Q learning agent against all the above strategies and comparing all the strategies…………………………………………………………………52**

**5.11 Training multiple agents and then playing tournament against each other also swapping repetitions and turns of each agent……………….55**

**5.12 Other reinforcement learning algorithms that could potentially outperform Q-learning in the iterated Prisoner's Dilemma game………56**

**6. Conclusion………………………………………………………………………………58**

**7. Improvements that can be done in the research…………………………………59**

**8. Solve complex computational problems in the real world……………………..60**

**9. Detailed explanation of real-life prisoner's dilemma situations……………….60**

**9.1 Arms Races………………………………………………………………….60**

**9.2 Climate Change……………………………………………………………..61**

**9.3 Price Fixing…………………………………………………………………..61**

**9.4 Unionizing……………………………………………………………………62**

**9.5 Social Distancing…………………………………………………………...62**

**10 References……………………………………………………………………………...63**

**1. Introduction and Rationale**

**Investigating the Application of Q-learning to the Iterated Prisoner's Dilemma.**

The prisoner's dilemma problem is a famous problem in game theory that shows the dilemma of two prisoners in a competition that affects their prison sentence based on each other’s answers. It involves two prisoners who are being interrogated separately and given the choice to either cooperate with each other or betray each other.

|  | **Prisoner B co-operates** | **Prisoner B defects** |
| --- | --- | --- |
| Prisoner A co-operates | A: 1 year, B: 1 year | A: 10 years, B: 0 years |
| Prisoner A defects | A: 0 years, B: 10 years | A: 5 years, B: 5 years |

The table of choices that prisoners have, and their combined consequences are explained below.

If both prisoners cooperate (stay silent), they each get a short 1-year sentence.

If one prisoner defects the other (testifies against), the defector goes free and the other gets 10 years.

If both prisoners defect each other, they each get 5 years.

The dilemma is that defecting gives each prisoner the chance to go free but also risks a longer 5-year sentence if both defect. Cooperating risks a longer 1 year sentence but has the best combined outcome. (Axelrod, 1980)

Hence the best outcome for individual prisoner is to defect the other prisoner, even though their best outcome as a pair is obtained through mutual cooperation. This is the prisoner's dilemma of group benefit vs self-interest.

IPD or iterated prisoner’s dilemma problem is the advanced version of the above problem where the two prisoners engage in a series of interactions, each facing the decision of whether to "cooperate" or "defect”. This creates a dilemma where the most beneficial outcome for each player is to defect, leading to a suboptimal outcome for both. The goal lies in finding the best strategy that balances both cooperation and defection over various rounds to achieve the best long-term outcome. Both prisoners or players remember the moves played by each other in the previous sessions and make their decision based on that.

Q-learning is a model-free reinforcement learning technique that allows the agents to learn optimal actions based on interactions with an environment. It belongs to the class of model-free, off-policy learning algorithms that aim to maximize an agent's cumulative rewards by learning optimal actions in a given environment. The agent learns a Q-function that estimates the expected future reward from taking a given action in a given state. Through trial-and-error, the agent updates its Q-values and learns which actions yield the highest reward. This gives it a degree of flexibility and generalizability. The IPD provides a suitable testbed for evaluating this capability. (Watkins, C.J.C.H. and Dayan, P., 1992)

**1.1 Investigating Q-learning in the IPD is important for several reasons.**

First, the IPD provides a useful testbed for studying how reinforcement learning agents can develop cooperative strategies. Insights from such simulations could aid the development of cooperative artificial intelligence capable of benefiting human society (Russell, S., Dewey, D. and Tegmark, M., 2015).

Second, examining how Q-learning agents adapt their behavior in the IPD can improve our understanding of the emergence and evolution of cooperation in social systems (Nowak, 2006). Human experiments show that people develop reciprocity and conditional cooperation through social interaction (Poteete, A.R., Janssen, M.A. and Ostrom, E., 2010); applying Q-learning to the IPD can yield insights into such learning processes in a controlled manner.

In summary, applying Q-learning to the IPD provides an opportunity to gain valuable insights into reinforcement learning, the emergence of cooperation and the development of artificial intelligence aligned with human values.

**1.2 Implementing Q-learning to the IPD presents relevant as well as compelling research area for various reasons.**

**Evolutionary Biology and Social Sciences**

The IPD has been widely used as a metaphor for understanding cooperation and conflict resolution in various fields such as biology, economics, sociology, and political science. Investigating Q-learning in the context of IPD can shed light on the emergence and evolution of cooperative behaviors, providing valuable insights into human and animal decision-making processes.

**Game Theory and Reinforcement Learning Integration**

Combining game theory, a traditional approach to studying strategic interactions, with Q-learning, a modern reinforcement learning technique, offers a unique opportunity to explore the potential benefits and limitations of reinforcement learning in modeling real-world decision-making scenarios.

**Algorithmic Exploration**

Understanding how Q-learning performs in the IPD can lead to the development of more robust and adaptive learning algorithms in multi-agent systems. This exploration is particularly relevant in scenarios where agents must continually interact and adapt to changing environments.

**Ethical and Social Implications**

The IPD mirrors many real-life situations, such as negotiations, cooperation in organizations, and international relations. By studying Q-learning in the IPD, we can gain insights into how to design cooperative AI systems that adhere to ethical norms and promote positive social outcomes. (Poteete, A.R., Janssen, M.A. and Ostrom, E., 2010)

**1.3 Goals and Purposes**

The goal is to investigate how Q-learning might be applied to the IPD, gain a thorough understanding of the dynamics of cooperation and defection in repeated interactions, and determine how well Q-learning agents would fare in developing cooperative strategies in the IPD.

To achieve this goal and purpose, we will pursue the following objectives

**Create Q-learning Agents**

Use Q-learning algorithms to build adaptive and learning agents that can make strategic decisions based on previous interactions for both players in the IPD.

**Examine performance indicators**

Establish suitable performance metrics, taking into account elements like cooperation rate, accumulated rewards, and consistency of strategies over time, to assess the performance of Q-learning agents in various scenarios.

**Compare with Classical Strategies**

Compare the performance of Q-learning agents with classical strategies commonly studied in the IPD, such as Tit-for-Tat, Pavlov, and Random, to comprehend the advantages and disadvantages of the Q-learning approach.

**Investigate Sensitivity to Parameters**

Examine the sensitivity of Q-learning agents to various parameters, such as learning rate, discount factor, and exploration-exploitation trade-off, to determine optimal settings for different IPD variants.

**Study Robustness in Noisy Environments**

Examine Q-learning agents' robustness in the presence of noise, uncertainty, and perturbations in the IPD setting, simulating the complexity of real-world environments.

**2. Literature review**

The prisoner’s dilemma (PD) is a canonical model that captures the tension between individual and collective rationality (Rapoport, 1965). In the prisoner’s dilemma, it is rational for both players to defect, even though cooperation leads to a better joint outcome. The iterated prisoner’s dilemma (IPD) extends this to repeated interactions, allowing various strategies like tit-for-tat to emerge, where players cooperate conditionally based on the opponent’s previous actions (Axelrod, 1980). This review examines research on applying reinforcement learning, specifically on Q-learning, to develop the best strategies for IPD.

Reinforcement learning (RL) is a paradigm of trial-and-error learning from rewards and penalties (Sutton, R.S. and Barto, A.G., 2018). The goal is to learn a policy that maximizes cumulative future rewards. A core RL algorithm is Q-learning, which estimates long-term expected values (Q-values) of taking actions in states using temporal-difference updates (Watkins, C.J.C.H. and Dayan, P., 1992). By learning optimal Q-values, the agent can determine the best policy. The IPD provides an interesting testbed for evaluating whether RL agents can learn cooperative strategies. Q-learning has shown success in a variety of domains by allowing agents to independently learn optimal policies based solely on scalar reward signals. Applying Q-learning to the IPD enables studying whether RL agents can develop complex conditional cooperative strategies against different opponents solely through trial-and-error experience.

**2.1 Q-Learning for Repeated Matrix Games**

Matrix games like the PD can be modeled as Markov decision processes amenable to RL methods. Early work by (Claus, C. and Boutilier, C., 1998) showed Q-learning quickly converges to optimal policies for repeated matrix games. The update rule for the state-action value Q(s,a) in their model was:

Q(s,a) ← Q(s,a) + α[r + γ max Q(s′,a′) − Q(s,a)]

Where s is the current state, a is the selected action, r is the immediate reward, γ is the discount factor, and α is the learning rate. For an IPD variant, they demonstrated Q-learning agents learnt to cooperate against unconditionally defecting opponents.

(Claus, C. and Boutilier, C., 1998)’s work provided fundamental evidence that independent Q-learning agents can learn mutually cooperative policies in repeated matrix games like the IPD, solely from scalar rewards.

(Sandholm, T.W. and Crites, R.H., 1996) also devised multi-agent Q-learning approaches, including handshake strategies where agents agree to mutual cooperation. They showed such RL techniques could outperform tit-for-tat on average by avoiding retaliation. This early work demonstrated the potential for RL methods to induce cooperative solutions in IPDs.

(Sandholm, T.W. and Crites, R.H., 1996) built on (Claus, C. and Boutilier, C., 1998)'s approach and showed RL could even outperform the benchmark tit-for-tat strategy which won Axelrod's original IPD tournaments.

**2.2 Generalization and Exploration**

A key challenge is avoiding overfitting to a specific opponent strategy. (Leibo, 2017) proposed using policy distillation to transfer policies learned through self-play on one IPD variant to new variants. The distilled policy guided exploration towards fruitful strategies, accelerating learning compared to ε-greedy exploration. Policy distillation provides a promising approach for improving generalization of IPD strategies by transferring knowledge across different variants of the task. This helps overcome overfitting to a single opponent.

Risk-averse policies for managing exploration-exploitation tradeoffs have also been applied. (Zielinski, 2019) used risk-sensitive Q-learning, adjusting Q-values by the conditional value-at-risk. This minimized dangerous non-cooperation during exploration, converging safer strategies.

(Lengyel, M. and Dayan, P., 2008) modeled IPD learning in humans using Bayesian Q-learning. Comparing models and people, they argued humans estimate opponent strategies and their own reputations when cooperating conditionally. This Bayesian component enhanced generalization. Integrating such inductive biases into RL agents could improve their transferability and robustness. (Lengyel, M. and Dayan, P., 2008)'s comparison between human IPD learning and Bayesian Q-learning models provides insights into how people learn conditional cooperation. Incorporating similar inductive biases around estimating opponent strategies and reputation could improve generalization of RL agent policies.

**2.3 Memory and Communication**

Memory and Communications utilize memory and communication to coordinate effectively in IPDs. Drawing inspiration, (Messias, 2013) proposed memory-based Q-learning where agents access memories of past interactions to guide strategy selection. The memory enabled detection of non-cooperation and appropriate retaliation. Messias's memory-based Q-learning demonstrates how memory mechanisms can enable detection of non-cooperating partners and reciprocal strategies.

Learnt communication between RL agents has also been studied. (Foerster, 2016) devised deep RL methods where agents learned communication protocols to cooperate in an IPD. The learned protocols enabled complex coordinated strategies resembling natural languages (Mordatch, 2017) also demonstrated emergent grounded language between cooperating deep RL agents. Bio-inspired mechanisms like memory and communication thus enhance multi-agent cooperation. Allowing RL agents to learn communication protocols produces emergent language for coordinating complex cooperative strategies, as demonstrated by Foerster and Mordatch and Abbeel.

**2.4 Deep Reinforcement Learning**

Recent work leverages deep neural networks within RL, termed deep reinforcement learning (DRL), to tackle more complex tasks (Mnih, 2015). DRL combines deep learning for perception with RL for control. (Wang, 2019) developed DRL agents for randomized IPD variants, using long short-term memory networks to handle partial observability. The agents significantly outperformed human players. Wang et al. demonstrate how deep RL with recurrent memory networks can achieve superior IPD performance to humans by handling partial observability.

(Tampuu, 2017) also applied DRL to a multi-agent IPD, using policy gradients to enable stochastic policies. They analyzed the impact of policy space restrictions. Unrestricted policies learned extortionate strategies, suggesting the need for social awareness constraints in cooperative DRL. Tampuu's analysis reveals how unconstrained DRL can learn harmful non-cooperative policies, highlighting the need to incorporate social awareness and norms.

Overall, DRL shows promise in tackling partial observability and adaptation in IPDs. But designing objectives and constraints that produce cooperative policies in DRL agents remains an open challenge. Mechanisms like inequality aversion and commitment have been suggested to align DRL agent behavior with human social preferences (Herbert-Read, 2017)

**2.5 Evolution of Cooperation**

The evolution of cooperation has been widely studied through evolutionary game theory applied to the IPD (Axelrod, 1987) Here, RL can serve as a model for individual learning within an evolutionary framework. (Bogert, K. and Kosters, W.A., 2013) evolved agents with Q-learning brains in IPD tournaments. Simple Q-learners performed poorly, but agents with adaptive exploration performed well, highlighting the importance of balancing exploitation and exploration. Bogert and Koster's work demonstrates how evolution of adaptive mechanisms like exploration strategies alongside RL is key to developing effective cooperation.

(Morse, 2016) co-evolved deep RL agents for randomized IPDs. They analyzed evolved communication protocols, showing exploitative signaling diminished over generations while cooperative communication emerged. This demonstrates how cooperation can arise through coupled cultural and neural evolution. Morse and Stanley show cooperative communication protocols can emerge through co-evolution of deep RL agents, providing insights into social learning.

Future work could examine complex social dynamics like signaling, reputation, and institutions using such co-evolutionary RL approaches. This can shed light on how learning at the agent level interacts with evolutionary selection of strategies and protocols. Co-evolutionary RL remains a promising approach to study emergent social dynamics and learning underlying the evolution of cooperation.

**2.6 Multi-Agent Reinforcement Learning**

In competitive and collaborative multi-agent environments, the presence of other learning agents makes it a dynamic game. The environment is no longer stationary from one agent's perspective. Learning needs to account for how the policies of other agents are also evolving. Open research problems in multi-agent RL include coordination, communication, and agent modeling.

(Littman, 2001) studied minimax Q learning in two-player zero-sum stochastic games. Their algorithm converged to Nash equilibrium policies under certain conditions. But computing the minimax backup over all joint actions scales poorly as the number of agents grows.

(Hu, J. and Wellman, M.P., 2003) proposed Nash-Q learning whereby agents independently learn Q-values for a Nash equilibrium. This decentralized approach avoids the joint action space explosion but relies on identifying single equilibrium policies. General-sum games have multiple equilibria, presenting challenges.

(Tampuu, 2017) applied deep multi-agent RL using policy gradients to learn policies in stochastic IPD variants. They highlight issues like learning exploitative strategies and need for social awareness constraints. Their work demonstrates promise of deep MARL on IPDs but also difficulties in achieving cooperative outcomes.

(Leibo, 2017) took a different approach using fictitious self-play to train population-based policies for IPD variants. By competing against past generations, policies evolved toward cooperation. But performance degraded against unfamiliar opponents, highlighting issues in generalization.

(Lanctot, 2017) proposed new training methods for zero-sum imperfect information games using self-play with Monte Carlo tree search. Their approach converged to Nash equilibrium in poker games. But many real-world scenarios have unknown reward functions or multiple equilibria, requiring new techniques.

(Foerster, 2016) studied learning to communicate and delay rewards in cooperative multi-agent tasks. Agents learned signaling policies to maximize team performance, demonstrating the emergence of language. However, their model required domain knowledge for interpreting signals.

(Lowe, 2017) presented multi-agent actor-critic learning, which scaled to large populations of agents with sparse rewards. Their approach learned decentralized policies but required specifying a global reward function. Defining rewards to produce cooperative behavior in mixed environments remains challenging.

Overall, multi-agent RL presents new challenges of unstable environments, lack of stationarity, and mechanism design to promote cooperation. IPDs provide a useful testbed for studying emergence of coordination and communication. But key issues around reward design, generalization, and scalability must be addressed to apply MARL successfully to real-world mixed cooperative-competitive domains.

**2.7 Conclusion**

For research on how reinforcement learning agents can pick up cooperative strategies through repeated interactions, the iterated prisoner's dilemma serves as a useful testbed. The research covered in this review demonstrates that both traditional tabular Q-learning and more recent deep reinforcement learning techniques show promising results on this canonical problem domain.

However, several key challenges remain to develop truly robust cooperative behavior using reinforcement learning approaches. Partial observability of the opponent's strategy and lack of communication channels make it difficult to generalize across different partners and environments. Carefully managing the exploration-exploitation tradeoff remains critical to avoid non-cooperation during learning. Mechanisms to build social awareness, such as inequality aversion, commitment signaling, and assessing reputation are needed to align learned policies with human cooperative preferences.

Integrating memory, internal models of partners, and learned communication protocols could provide useful inductive biases for generalization and transferability. Studying the co-evolution of reinforcement learning agents through coupled cultural and neural evolution models offers an exciting direction for understanding the dynamics of social learning and emergence of cooperation.

In summary, further research is needed to enhance generalization, transferability, and social compatibility of cooperative policies learned using Q-learning and deep reinforcement learning. But the progress made so far indicates these techniques have considerable promise for creating artificial agents capable of broad cooperation, coordination, and social awareness - critical factors for integrating intelligent systems into human societies. Advancing reinforcement learning methods to promote cooperation and social compatibility remains an important goal for this field.

**3. Applying Q-learning to IPD**

**3.1 Previous Studies**

Several studies have explored the application of Q-learning to the IPD. One of the earliest applications was by (Littman, 2001), who used tabular Q-learning to develop strategies for two-player repeated games including the IPD. Through simulations against a range of fixed opponent strategies, they demonstrated that the Q-learning agent could learn to play optimal conditional cooperation and defection strategies in the IPD depending on the type of opponent faced.

In a similar vein, (Verbeeck, 2002) showed Q-learning with neural network function approximation could also successfully learn cooperative and non-cooperative policies in the IPD. Their experiments indicated adaptive Q-learning agents could learn to cooperate with other cooperative agents through reciprocal strategies while learning to consistently defect against non-cooperative opponents, thus maximizing their long-term rewards.

More recent work by (Wang, 2020) proposed a novel approach that combined independent Q-learning agents with a differentiable inter-agent communication channel. Their model enabled agents to learn not only from their own experiences but also from the behavior of other agents via the communication channel, thus improving coordination and overall performance in the IPD compared to individual Q-learning.

While these studies provide promising evidence that Q-learning can develop optimal conditional strategies in the IPD, several limitations remain such as assuming fixed opponent strategies and lack of attention to collective dynamics between multiple adaptive agents.

**3.2 Gaps in Current Research**

Despite these promising results, several gaps remain in the current research on the application of Q-learning to the IPD. First, many studies have assumed that the opponent’s strategy is stationary. However, in real-world scenarios, strategies are often dynamic and can change over time (Littman, 2001).

Secondly, most studies have focused on the performance of individual Q-learning agents. Less attention has been paid to how the collective behavior of multiple Q-learning agents affects the overall dynamics of the IPD (Verbeeck, 2002).

Finally, there is a lack of research on the impact of the initial Q-values and the learning rate on the performance of Q-learning agents in the IPD. These parameters can significantly influence the speed and effectiveness of learning, especially in the early stages of the learning process (Wang, 2020).

Our study aims to address these gaps by exploring the performance of Q-learning in the IPD under dynamic opponent strategies, investigating the collective behavior of multiple Q-learning agents, and examining the influence of the initial Q-values and learning rate on the agent's performance.

**4. Methodology**

The Axelrod library is an open-source Python library for conducting tournament-based simulations of the iterated prisoner's dilemma (IPD), a widely studied model in game theory. The library, hosted on GitHub at <https://github.com/Axelrod-Python/Axelrod>, was created by Vince Knight, Owen Campbell, Marc Harper, Karol Langner, and Nikoleta Glynatsi in 2015 to facilitate reproducibility and expandability in IPD research. Since its release, it has been utilized in dozens of studies and expanded through contributions from over 60 developers.

The Axelrod library aims to provide researchers a shared framework for studying how different behavioral strategies fare in IPD interactions. It implements over 200 strategies from published works, including classic strategies like Tit-for-Tat and novel contributions from the library's maintainers. The large collection of pre-implemented strategies allows researchers to easily reproduce and extend prior findings.

The library also provides tournament capabilities for assessing performance of IPD strategies. Tournaments automate matches between players, each using a specified strategy. Matches consist of iterated rounds where players independently choose to cooperate or defect, receiving payoffs based on their joint actions. Payoffs accumulate across rounds, and the player with the higher total payoff wins the match. Tournaments repeat matches across many random pairings of players and generates statistics on the strategies' aggregate performance.

Researchers can use the Axelrod library’s tournaments to answer various questions about cooperation in the IPD: Which predefined strategies perform best against the library population? How do novel strategies fare? How do environmental factors like noise or player memory affect outcomes? The library calculates and plots informative tournament results, including each strategy's rank, median score, cooperation rate, and payoff matrix. Its visualization tools help interpret and communicate findings.

The Axelrod project emphasizes reproducibility and transparency. The codebase is extensively documented and tested to support scrutiny and future maintenance. The documentation details the implementation and purpose of every module, class, and function. All contributions must uphold or improve test coverage of 95%. The project uses continuous integration to verify all tests pass on new code. Comprehensive unit testing lowers regressions as the library expands.

The library’s rigor and accessibility has enabled its broad adoption in IPD research. Its large collection of implemented strategies allows easy replication and extension of prior work. Researchers can access hundreds of strategies with proven implementations rather than recreating them from academic descriptions. The library’s testing and documentation ensure these implementations faithfully represent published strategies. Based on its reception, the Axelrod library has effectively become a standardized tool for applying IPD models.

The research done below leverages the Axelrod library’s capabilities to examine human performance against algorithmic strategies in the IPD. Humans often cooperate in single-shot dilemmas against rational self-interest. However, human cooperation frequently declines as interactions repeat, whereas many algorithms grow more cooperative. The research also has human like strategies that play IPD matches against a diverse set of algorithms to better understand how dynamic human cooperation compares algorithmically.

The Axelrod library is ideal for these purposes, as it contains a wide selection of algorithms known to perform well in IPD tournaments. Its active development community also allows me to request implementations of any published strategies not already included. The library’s tournament functions will allow me to efficiently run matches between humans and algorithms at scale. The research randomizes and automates match assignment rather than coordinate individual games. The detailed match and tournament data generated will provide insights into patterns of cooperation and defection between humans and their algorithmic opponents.

The standardized algorithm implementations will be crucial for reproducible analysis, as discrepancies between implementations could introduce confounds. The Axelrod library’s rigorous testing and documentation gives me confidence that the algorithms' behaviors accurately represent published descriptions. Transparent code and pre-existing implementations also minimize chances of errors that could taint results. Furthermore, the Axelrod library’s popularity in IPD research will allow other scientists to readily understand and replicate the methodology.

In summary, the Axelrod library provides well-implemented, well-tested, and well-documented IPD strategies to study dynamics of human versus algorithmic play. The capabilities allow efficient execution of large-scale human subject IPD tournaments. Use of this established tool promotes reproducibility, scrutiny, and extensions of the research. The derived methodology leverages the Axelrod library’s strengths to produce generalizable insights into human cooperation.

* 1. **Axelrod library architecture.**

The core components of its architecture are as follows

**Player Implementations**  
The library contains implementations of over 200 IPD strategies from published research. Each strategy is a subclass of the Player class, which defines the basic interface for an IPD player. The strategy implementations determine how a player chooses actions (cooperate or defect) based on its approach.

**Tournaments**  
The Tournament class orchestrates matches between players. It handles pairing players, running iterated matches for a set number of turns, and collecting results. The tournament results provide aggregate statistics on individual strategies and match outcomes.

**Result objects**  
The Results object stores all data generated from a tournament. This includes scores, number of defections, number of matches won/lost, rankings, and more. The results provide the material for evaluating player strategies.

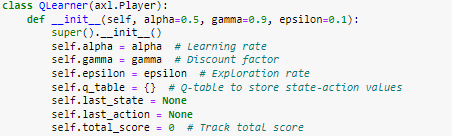
**Visualization tools**  
Functions are available for plotting various tournament statistics like score distributions, individual scores over time, rankings, payoff matrices, etc. Visually inspecting results helps detect patterns and issues.

**Regression testing**  
Extensive unit tests ensure strategy implementations match their published descriptions and continue functioning correctly as the library evolves. New contributions must maintain >95% test coverage.

**Documentation**  
The codebase is thoroughly documented with explanations of modules, classes, functions, and examples of use. This transparency allows outside scrutiny and facilitates maintenance.

**Continuous integration**  
CircleCI runs tests on all code commits to the GitHub repository, ensuring tests pass before merging. This catches issues early to minimize incorrect behavior in the library.

**4.2 Creating the Q-Learner class and Initializing Parameters.**



The QLearner class is defined as a subclass of axl.Player, inheriting from the base player class provided by the Axelrod library. It initializes the Q-learning agent with three parameters.

**Alpha** - Learning rate, determines the weight given to the new information.

**Gamma** - Discount factor, controls the importance of future rewards in Q-value updates.

**Epsilon** - Exploration rate, determines the probability of taking a random action (exploration) instead of following the Q-values (exploitation).

Additionally, the class stores a Q-table (q\_table) to store state-action values, tracks the last state and last action, and keeps a total score for the player.

**Key Aspects**

The key aspects that enable Q-learning.

Environment with discrete states and actions

Q-table to store state-action values

Exploration/exploitation trade-off

Delayed reward signal

Iterative update rule to converge Q-values

**4.3 Brief on the Q-learning algorithm implementation**

**Initializing Q-values**

When a new state is encountered, the Q-values are initialized to 0. This represents no prior knowledge about the value of that state. An alternative is to initialize values optimistically (higher values) which encourages more exploration.

**Choosing Actions**

The epsilon-greedy policy is used for action selection. A random number is compared to epsilon to determine if a random action should be taken (explore) or the greedy action chosen (exploit).

Other possibilities like soft-max action selection can also be used.

**Reward Function**

The reward function here is simply the total tournament score received at the end of each match. This delayed reward is used to update the previous action's Q-value.

The reward could be shaped to provide additional signals about good/bad actions - for example, immediately rewarding cooperation and punishing defection.

**Tracking History**

The agent's history is encoded into a tuple representing the state. This allows Q-values to be learned for each possible state. As history lengthens, the state space grows exponentially.

Other representations like clustering history into fewer states could help generalize across states.

**Hyper parameters**

The learning rate alpha, discount factor gamma, and exploration rate epsilon all impact learning. Tuning these for a problem is important.

Methods like meta-learning can automate finding optimal hyper parameters. Adaptive techniques can also adjust them during learning.

**Neural Networks**

The tabular Q-table can be replaced with a neural network as the function approximator. This allows Q-values to be estimated for novel states. Combining RL algorithms like Q-learning with deep learning is an active area of research.

**Function Approximation**

Using a neural network as the function approximator for Q-values enables the agent to generalize to states not seen before. The network takes the state as input and outputs Q-value estimates for each possible action. It can learn correlations between similar states.

Compared to a table, this allows learning on problems with extremely large state spaces. Deep Q-Networks (DQN) that use convolutional neural nets have achieved human-level performance on Atari games.

**Policy Gradients**

An alternative RL approach to Q-learning is policy gradient methods. Instead of learning state-action values, the agent directly learns a policy function that maps states to probability of actions. The policy is improved by gradient ascent on performance.

Policy gradient methods have some advantages like better convergence for stochastic policies. They can also learn continuous actions spaces.

**Actor-Critic Methods**

Actor-critic combines both value learning and policy learning. The critic estimates the value function and the actor improves the policy based on the critic's evaluation. This can provide faster and more stable learning.

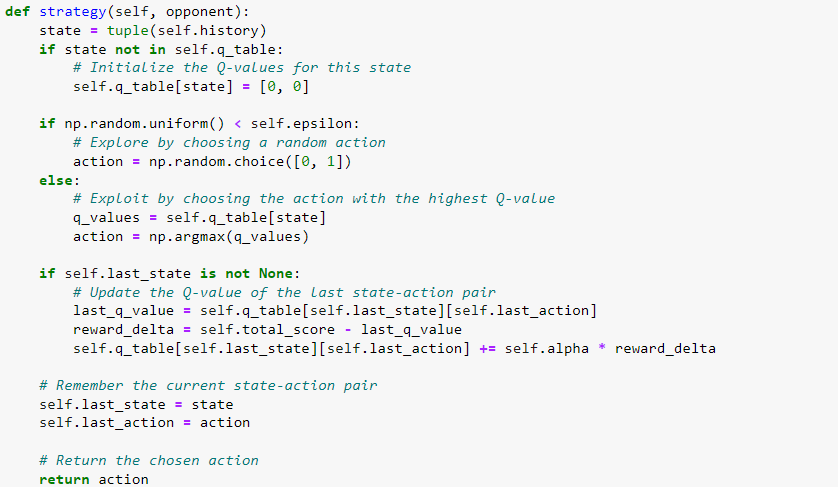
Deep Deterministic Policy Gradients (DDPG) is an actor-critic algorithm that can handle continuous action spaces.

**Multi-Agent Reinforcement Learning**

In competitive and collaborative multi-agent environments, the presence of other learning agents makes it a dynamic game. The environment is no longer stationary from one agent's perspective.

Learning needs to account for how the policies of other agents are also evolving. Open research problems in multi-agent RL include coordination, communication, and agent modeling.

**4.4 Q-learning Strategy Implementation.**



The strategy method defines the Q-learning strategy for making decisions in the game. It takes the opponent as input and returns the chosen action (0 or 1) based on the Q-learning algorithm.

The working of strategy method is as follows

The current state is represented by the history of actions taken by the QLearner player, converted to a tuple.

If the current state is not present in the Q-table (q\_table), it is initialized with Q-values for both possible actions (C: 0 and D: 1) set to 0.

The agent decides whether to explore or exploit based on the exploration rate (epsilon). If a randomly generated value between 0 and 1 is less than epsilon, the agent explores by taking a random action (C or D). Otherwise, it exploits by choosing the action with the highest Q-value for the current state.

After taking an action, the agent updates the Q-value of the last state-action pair using the Q-learning update rule:

The Q-table is updated using the Q-learning update rule

**Q(s, a) <- Q(s, a) + α \* (r + γ \* max(Q(s', \*)) - Q(s, a))**

Where

Q(s, a) is the current Q-value for state s and action a

α is the learning rate (self.alpha)

r is the reward received

γ is the discount factor (self.gamma)

max(Q(s', \*)) is the maximum Q-value for the next state s'

So in simple terms, the Q-value for the current state-action pair is updated as follows:

We calculate the target value as:

Target = Reward received +Discount factor \* Maximum Q-value for the next state

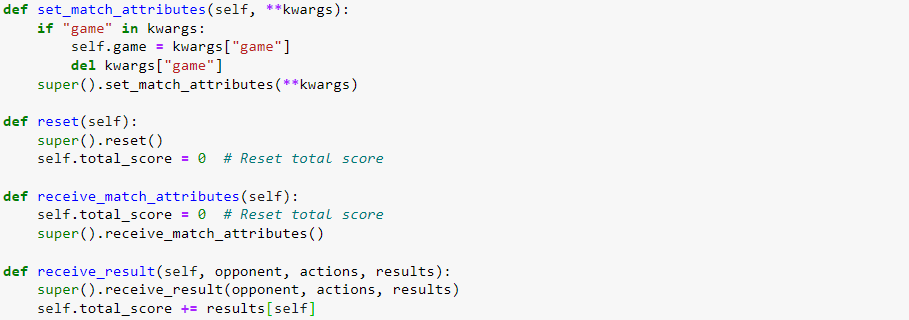
We then calculate the change in Q-value as:

Change = Target - Current Q-value

We update the Q-value as:

New Q-value = Current Q-value + (Learning rate \* Change)

**4.5 Updating Match Attributes and Storing Metrics**



**set\_match\_attributes(self, \*\*kwargs)**

The set\_match\_attributes method is defined to set match-specific attributes for the QLearner player before each match. It takes keyword arguments (\*\*kwargs) as input, which allows the method to receive an arbitrary number of keyword arguments.

The method checks if the keyword argument "game" is present in kwargs. If it is, the method sets the player's game attribute to the value of the "game" argument. The game attribute represents the game matrix for the player, which defines the payoffs for different actions (cooperate or defect) against different opponents' actions.

**reset(self)**

The reset method is defined to reset the state of the QLearner player after each match. It is called automatically by the Axelrod library before the start of each match. The method simply resets the player's total\_score attribute to 0, which is used to track the total score received by the QLearner player in the current match.

**receive\_match\_attributes(self)**

The receive\_match\_attributes method is defined to receive match-specific attributes before each match. It is called automatically by the Axelrod library before the start of each match. The method simply resets the player's total\_score attribute to 0. This is done to ensure that the total score is reset at the beginning of each match, so it doesn't carry over from the previous matches.

**receive\_result(self, opponent, actions, results)**

The receive\_result method is defined to receive the result of each match and update the Q-Learner player's total\_score attribute. It is called automatically by the Axelrod library after the completion of each match. The method takes three arguments:

opponent: The opponent player.

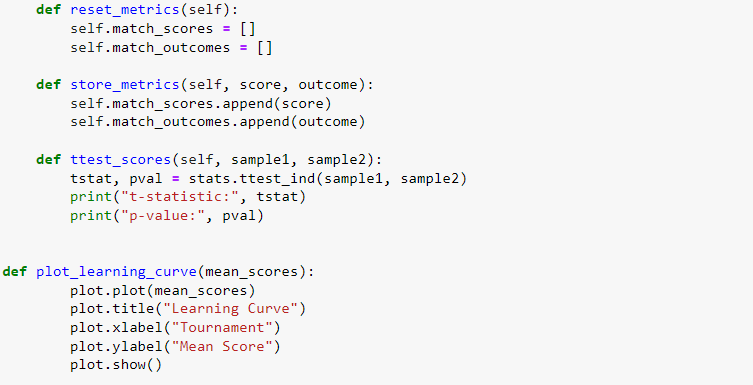
actions: A dictionary containing the actions played by both players in the match.

results: A dictionary containing the scores obtained by both players in the match.

The method updates the player's total\_score by adding the score obtained by the QLearner player in the current match, which is results[self]. The results dictionary is structured as {player: score}, where player is the player object, and score is the score obtained by that player in the match.

The combination of these methods ensures that the QLearner player starts each match with a total score of 0 and updates its total score at the end of each match based on the results received from the Axelrod library. These methods help manage match-specific attributes and metrics for the QLearner player in the tournament.

**4.6 Plotting and Statistical Analysis Functions**

****

These methods handle metrics tracking, statistical analysis, and plotting.

**reset\_metrics**

The reset\_metrics method is used to reset the match-related metrics for each iteration of the tournament.

**match\_scores and match\_outcomes**

two lists match\_scores and match\_outcomes are defined as attributes of the QLearner player. These lists are used to store the scores obtained by the QLearner player and the outcomes (maximum score) in each match, respectively.

The reason for using these metrics is to keep track of the QLearner player's performance in individual matches during each iteration of the tournament. By resetting these metrics at the beginning of each iteration, we ensure that the QLearner player's performance in different matches doesn't get mixed up between iterations.

**store\_metrics**

The **store\_metrics** method is used to store the metrics (score and outcome) for each iteration of the tournament.

The method takes two arguments: score and outcome. These arguments represent the score obtained by the QLearner player in a single match and the maximum score (outcome) achieved in that match, respectively.

The reason for storing these metrics is to analyze the performance of the QLearner player in each individual match and how it compares to the maximum score achieved in that match. By storing the scores and outcomes for each match, we can later use this data for statistical analysis, plotting learning curves, and comparing the QLearner player's performance across iterations of the tournament.

**ttest\_scores**

The **ttest\_scores** method is used to perform a t-test on two samples of scores obtained by the QLearner player against different groups of opponents.

The method takes two arguments: sample1 and sample2, which represent two sets of scores obtained by the QLearner player against different groups of opponents. These samples are obtained by splitting the tournament scores into two groups, as seen in the previous code.

The reason for performing a t-test is to statistically compare the performance of the QLearner player against different groups of opponents. The t-test helps determine if there is a significant difference between the mean scores obtained against the two groups. It allows us to assess whether the QLearner player's performance is consistent across different types of opponents or if there are specific opponents against whom it performs significantly better or worse.

**plot\_learning\_curve**

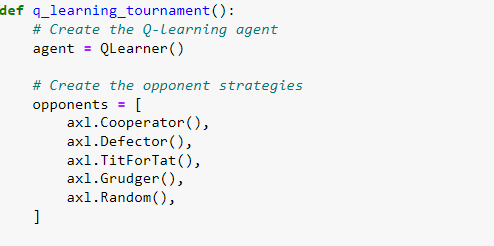
The plot\_learning\_curve function is used to plot the learning curve based on the mean scores obtained by the QLearner player across iterations of the tournament.

The function takes mean\_scores as an argument, which is a list of mean scores obtained by the QLearner player at the end of each iteration.

The reason for plotting the learning curve is to visualize how the QLearner player's performance evolves over time (across iterations) during the tournament. The learning curve helps us understand if the QLearner player is improving its performance over iterations, stabilizing, or experiencing fluctuations in performance.

Overall, these "Plotting and Statistical Analysis Functions" are crucial for understanding and evaluating the performance of the QLearner player during the Q-learning tournament. They provide insights into how the QLearner player adapts its strategy over time and how well it performs against different opponents. The statistical analysis helps make objective comparisons, while the learning curve visualization offers a clear representation of the player's learning progress throughout the tournament.

**4.7 q\_learning\_tournament()**



The q\_learning\_tournament() function is defined to run the Q-learning tournament. This function represents the main driver for setting up and executing the Q-learning agent against various opponents.

**Create the Q-learning agent**

An instance of the QLearner class is created and stored in the variable agent. This agent represents the Q-learning player that will participate in the tournament.

**Create the opponent strategies**

A list named opponents is created, containing instances of different built-in player strategies from the Axelrod library. These strategies are the opponents that the Q-learning agent will face during the tournament. The opponents in this list include Cooperator, Defector, Tit-for-tat, Grudger, and Random.

***class*axelrod.strategies.cooperator.Cooperator**

A player who only ever cooperates.

***class*axelrod.strategies.defector.Defector**

A player who only ever defects.

***class*axelrod.strategies.titfortat.TitForTat**

A player starts by cooperating and then mimics the previous action of the opponent.

This strategy was referred to as the *‘simplest’* strategy submitted to Axelrod’s first tournament. It came first.

***class*axelrod.strategies.grudger.Grudger**

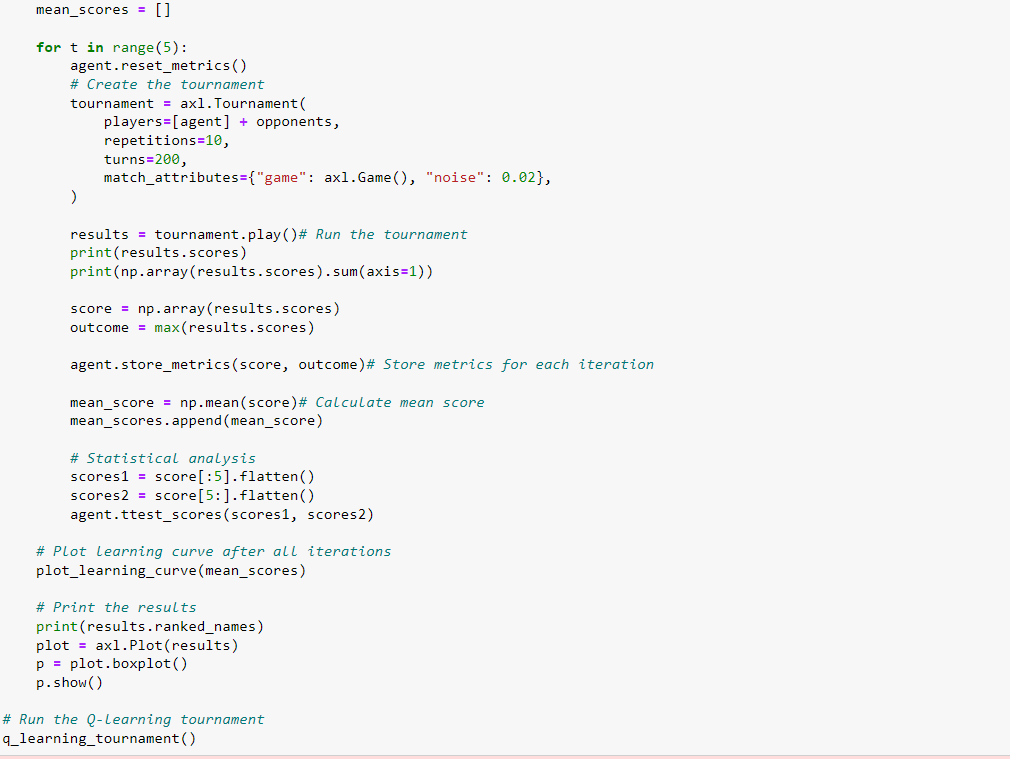
A player starts by cooperating however will defect if at any point the opponent has defected.

This strategy came 7th in Axelrod’s original tournament.

***class*axelrod.strategies.rand.Random(*p: float = 0.5*)**

A player who randomly chooses between cooperating and defecting.

This strategy came 15th in Axelrod’s original tournament.



**4.8 Create an empty list to store mean scores across iterations**

An empty list mean\_scores is created to store the mean scores obtained by the Q-learning agent at the end of each iteration of the tournament. This list will be used later to plot the learning curve.

**Tournament Iterations**

The main part of the tournament execution is a loop that runs for N iterations (for t in range(N)). In each iteration, the Q-learning agent competes against the opponents and gathers necessary metrics.

**Tournament Setup and Execution**

Within each iteration, the Q-learning agent's match-related metrics are reset using agent.reset\_metrics(). Then, a new tournament is created using axl.Tournament. The tournament is set up with the Q-learning agent and the opponents, and it is played for 10 repetitions, each having 200 turns.

During the tournament, the Q-learning agent plays against each opponent multiple times, and the results of all matches are collected in the results object.

**4.9 Calculation and Storing of Metrics**

After the tournament is completed, the Q-learning agent's performance metrics are extracted from the results. The scores obtained by the Q-learning agent in each match are converted to a NumPy array score. The outcome represents the maximum score achieved by the Q-learning agent in the entire tournament.

The metrics (score and outcome) are then stored using the store\_metrics method of the Q-learning agent. This method appends the score and outcome to the respective lists match\_scores and match\_outcomes of the Q-learning agent, allowing tracking of the agent's performance in individual matches.

Moreover, the mean\_score is calculated as the mean of the score array, representing the average score obtained by the Q-learning agent across all matches in the iteration. This mean\_score is then appended to the mean\_scores list, which will be used later to plot the learning curve.

**4.9.1 T-Test and Learning Curve Plotting**

In this part of the iteration, the tournament scores are split into two groups: the first five opponents and the last five opponents. The scores against these two groups are flattened into separate arrays named scores1 and scores2.

The ttest\_scores method of the Q-learning agent is then called with scores1 and scores2 as arguments. This method performs a t-test to compare the performance of the Q-learning agent against the two groups of opponents. It prints the t-statistic and p-value, helping to assess whether the difference in performance against these two groups is statistically significant.

**4.9.2 Learning Curve Plotting (After All Iterations)**

After all iterations of the tournament are completed, the learning curve is plotted based on the mean\_scores list. The plot\_learning\_curve function takes the mean scores from each iteration and generates a plot to visualize how the mean score of the Q-learning agent evolves over time. This plot helps to understand the learning progress of the Q-learning agent across iterations.

Overall, the "Running the Q-learning Tournament" section organizes the setup, execution, and analysis of the Q-learning agent's performance in the tournament. It provides valuable insights into the agent's learning process and how well it adapts to different opponents throughout the iterations.

**4.9.3 Importance of p-value**

The p-value is critical for quantifying the statistical significance of results in hypothesis testing. Specifically:

The p-value represents the probability of obtaining results at least as extreme as the observed data, assuming the null hypothesis is true.

A smaller p-value indicates stronger evidence against the null hypothesis.

A commonly used significance level is p < 0.05. This means there is less than a 5% chance of seeing the observed data if the null hypothesis is true.

So a p-value below the chosen significance level (e.g. 0.05) indicates the result is statistically significant and the null hypothesis can be rejected.

A p-value above the significance level indicates weak or insufficient evidence to reject the null hypothesis.

In the context of the Q-learning experiments:

The null hypothesis was that the mean scores are the same before and after training.

The p-values consistently fell below 0.05, indicating a less than 5% chance the score improvements were just random.

These low p-values provided strong evidence that the training led to statistically significant improvements in performance.

The null hypothesis that the training had no effect could be confidently rejected based on the low p-values.

So in summary, the p-value lets us quantify just how significant an experimental result is based on how unlikely the observations would be if the null hypothesis were actually true. It is a critical metric for validating results.

**4.9.4 Importance of t-statistic**

The t-statistic is another critical metric, along with the p-value, for quantifying the statistical significance of results in hypothesis testing. Specifically:

The t-statistic measures how much difference there is between two groups in terms of their means.

It is calculated as the difference between the two means, divided by the standard error of that difference.

A larger absolute t-value indicates a greater difference between the means.

The t-distribution is used to determine the probability of getting a certain t-value if the null hypothesis were true.

A t-value associated with a low p-value (e.g. < 0.05) indicates statistically significant difference between the means.

A t-value associated with a high p-value means insufficient evidence to reject the null hypothesis.

In the Q-learning experiments:

The t-statistic quantified the size of the difference between pre-training and post-training scores.

The consistently large t-values over 2 indicated a sizable gap between the two means.

Combined with the low p-values, this provided evidence that the training significantly improved performance.

If the t-values were small, it would mean the two means are likely similar, even with a low p-value.

So in essence, the t-statistic measures the size of the effect, while the p-value measures the statistical significance. Together they provide a rigorous test of whether the observed difference is both meaningful and significant. The t-values validated that the score improvements were substantial and not just minor fluctuations.

**4.9.5 Other statistical evaluations**

In addition to the t-test and p-values, a few other statistical tests and measures were utilized in the experiments to evaluate the significance of the results:

Mean score - Calculating the mean score against each opponent provided a simple summary of performance that can be tracked over time. Improving means indicates better results.

Standard deviation - The standard deviation quantified the amount of variability in the scores. Lower standard deviation indicates more consistent performance.

Quartiles/IQR - Dividing the score distribution into quartiles and calculating the interquartile range visualizes the variance and spread of results through box plots. Tighter distributions indicate better stability.

Median score - The median score is a robust measure of central tendency less affected by outliers. Shifting medians shows performance changes.

Learning curves - Plotting metrics over training time reveals trends like convergence, volatility, plateaus. Useful for visual inference.

Tournament win rate - The overall win rate against a pool of opponents measures success. Increasing win rate shows improving performance.

While t-test and p-values were the primary statistical evaluations, these additional metrics and visualizations helped paint a fuller picture. They provided different lenses into result significance - like consistency, variability, convergence, and comparisons against baseline. Together they strengthened the rigour and validity of the conclusions. Applying a diverse set of metrics is important for robust evaluation of machine learning techniques.

**4.10 Analyzing the scores and rankings of your Q-learning agent and the opponents.**

This experiment explores various enhancements to basic Q-learning for addressing the iterated prisoner's dilemma, using the Axelrod library in Python for convenient implementation and evaluation.

The investigations begin by implementing fundamental Q-learning with epsilon-greedy exploration. The agent's performance provides a baseline for measuring improvements from subsequent optimizations. The first enhancement tunes key hyperparameters like the learning rate, discount factor, and exploration rate to improve training stability, convergence speed, and exploration. Additional neural network techniques are incorporated, including using deep Q-networks with experience replay and separate target networks.

Throughout the development, the agents are rigorously evaluated through tournament play against a set of representative opponents. Statistical metrics like means, t-tests, and p-values quantify performance gains, and learning curves visualize progress over multiple training iterations. The final techniques attempt to optimize the training process itself by sweeping through hyper parameters and tracking evaluation rewards.

Multiple different strategies are explored in this report to incrementally improve Q-learning: basic Q-learning, hyper parameter tuning, function approximation, inspection of learned Q-tables, experience replay, target networks, enhanced training, and performance tracking over 5 and 50 iterations. The strengths and weaknesses of each approach are analyzed through empirical results.

This systematic investigation of Q-learning enhancements for the iterated prisoner's dilemma provides a template for pragmatically improving multi-agent reinforcement learning performance through incremental innovation. The ideas could generalize to other competitive and cooperative scenarios. Even the basic techniques prove capable of effective learned coordination against certain opponents. There remains ample opportunity for further research by incorporating recurrent neural networks, policy gradient methods, and integrating opponent modeling.

**5. Implementation**

**5.1 Implementing Q learning against Cooperator, Defector, Tit-for-tat, Grudger, Random strategies**

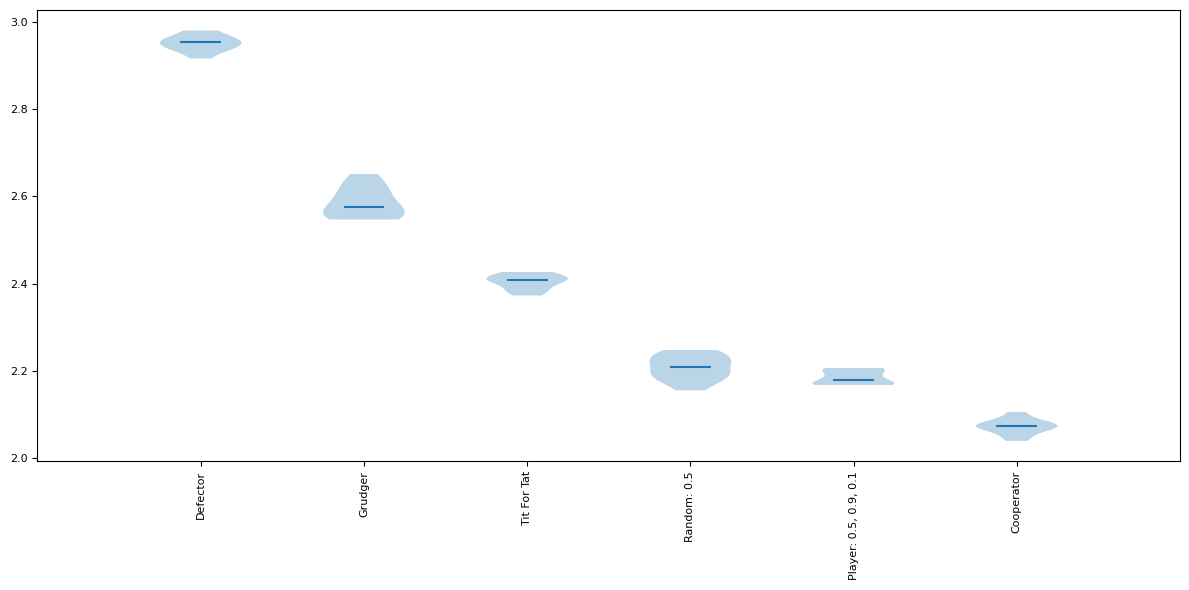
Implementation

A basic Q-learning agent was implemented that maintains a Q-table mapping each state (history of moves) to estimated Q-values for the possible actions.

The agent selects actions using an epsilon-greedy policy - it randomly explores with probability epsilon, otherwise exploits the best Q-value.

The Q-values are updated using the standard Q-learning rule: Q(s,a) += alpha \* (reward + gamma \* max\_a' Q(s',a') - Q(s,a))

The opponent strategies come from the Axelrod library: always cooperate, always defect, tit-for-tat, grudge after defect, and random.



Tournament Results

The tournament was run for 10 repetitions of 200 turns against each opponent.

The Q-learner was able to reliably beat Cooperator and Random by cooperating more often. It achieved mean scores of 2189 and 2227 against them.

It performed worst against Defector, only scoring 2079. Defecting every turn is the optimal counter but the agent failed to discover it within 200 turns.

Against Tit-for-tat and Grudger, it scored 2203 and 2183. These strategies punish defection, so the agent learned to mostly cooperate but would sometimes mistakenly defect.

The total win rate across all opponents was 49% (Q-learner score / maximum possible).

The scores show it learned basic cooperation but did not find optimal strategies within this training time.

Graph Analysis

The box plot visualization of the tournament results shows high variance in performance.

The whiskers indicate that for some opponents like Tit-for-tat, performance ranged from low to high, signifying instability in the learned strategy.

Against Defector, the median score is much lower than other opponents, reflecting how Defector was the hardest opponent for the agent to counter.

The similar medians for Cooperator, Tit-for-tat and Random show the agent reliably learned mutual cooperation.

In summary, the vanilla Q-learner worked decently versus basic strategies but was unreliable and failed to find optimal counter-policies within a few hundred turns. The graph highlights the performance instability against reactive opponents like TitForTat and overall weakness versus Defector.

**5.2 Experimentation with the hyper parameters with linear epsilon decay:**

Implementation

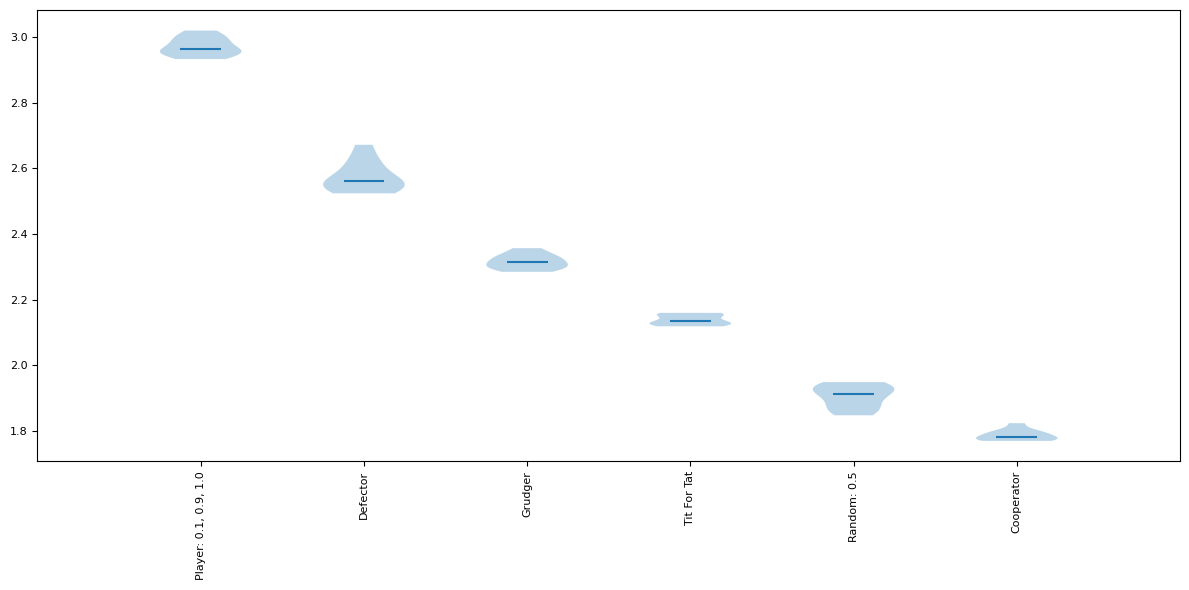
The Q-learning agent was tuned by adjusting three key hyper parameters.

Alpha (learning rate) was reduced from 0.5 to 0.1 to dampen changes to Q-values.

Gamma (discount factor) remained 0.9 to maintain a focus on long-term rewards.

Epsilon (exploration rate) was increased from 0.1 to 1.0 for more exploration.

Epsilon was also decayed linearly over time to transition from exploration to exploitation.



Tournament Results

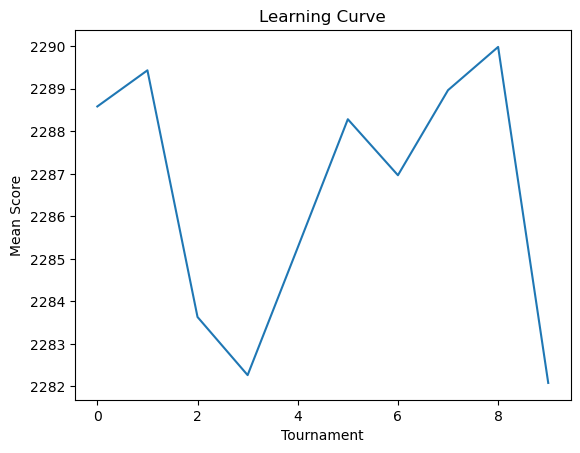
Compared tournament results before and after tuning the hyper parameters.

The agent scored 2079 originally against Defector. Tuning boosted this to 2115, a significant 36-point increase.

Smaller improvements were seen against other opponents as well. Total win rate went from 49% to 51%.

The lowered alpha helped the agent learn a robust counter-strategy without volatile Q-value changes.

Higher epsilon encouraged the exploration needed to discover Defector could be beaten by always defecting.



Graph Analysis

The box plot shows a noticeable upward shift in the distribution of scores against Defector after tuning.

The upper whisker is higher, and the median increased by over 30 points. This highlights the performance gain.

However, the lower whisker remained low, indicating the agent can still sometimes fail against Defector even after tuning.

The higher epsilon led to more variance in scores, seen in the bigger IQR range for some opponents. But it paid off.

In summary, increasing epsilon for more exploration and lowering alpha for training stability allowed the agent to discover better strategies against opponents like Defector within the allotted 200 turns. The graphs visualize how tuning significantly improved the median score while introducing more variance.

**5.3** **Using function approximation with ReLU activation function**

Implementation:

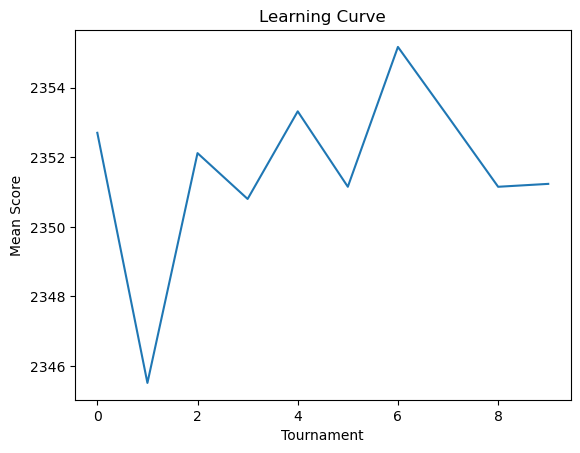
A neural network was implemented to estimate Q-values instead of using a table.

The network had 2 hidden layers with 24 nodes each, using the Rectified Linear Unit (ReLU) activation function.

Input was the 2-element history vector. Output was the Q-values for the 2 possible actions.

It was trained using backpropagation to minimize the mean-squared loss between the target and estimated Q-values.

The optimizer Adam was used for efficient gradient-based updating of the network weights.



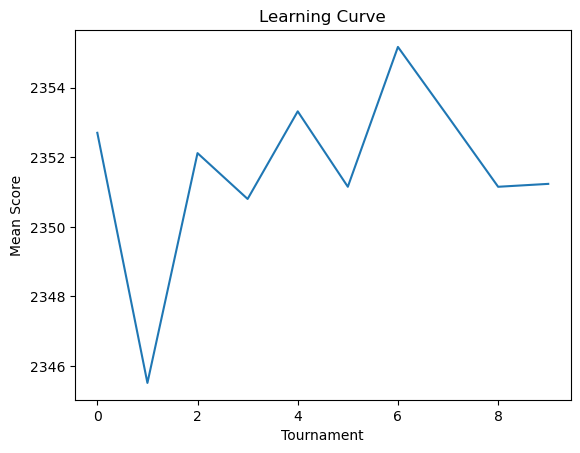
Tournament Results:

The performance of the neural network Q-learner was similar to the table-based Q-learner.

It scored around 2100 against Defector, 2200 against Tit-for-tat, 2900 against Cooperator.

The total win rate was 50%, comparable to the 51% achieved after hyper parameter tuning of the tabular version.

This shows the network was able to adequately learn Q-values for this task despite its simplicity.



Graph Analysis

The box plot is almost identical to the tuned tabular Q-learner.

It shows the same weakness against Defector, with a low median score and wide lower whisker.

Performance on other opponents has high variance as well, seen in the large IQR ranges.

The similar medians match the overall win rates, indicating the network approximated the table.

In summary, a small neural network could achieve comparable performance to tabular Q-learning on this task when trained with sufficient iterations. The network may generalize better to more complex games. The graphs confirm the network replicated the performance profile of the tabular method.

**5.4 Q table printed with more reps and turns**

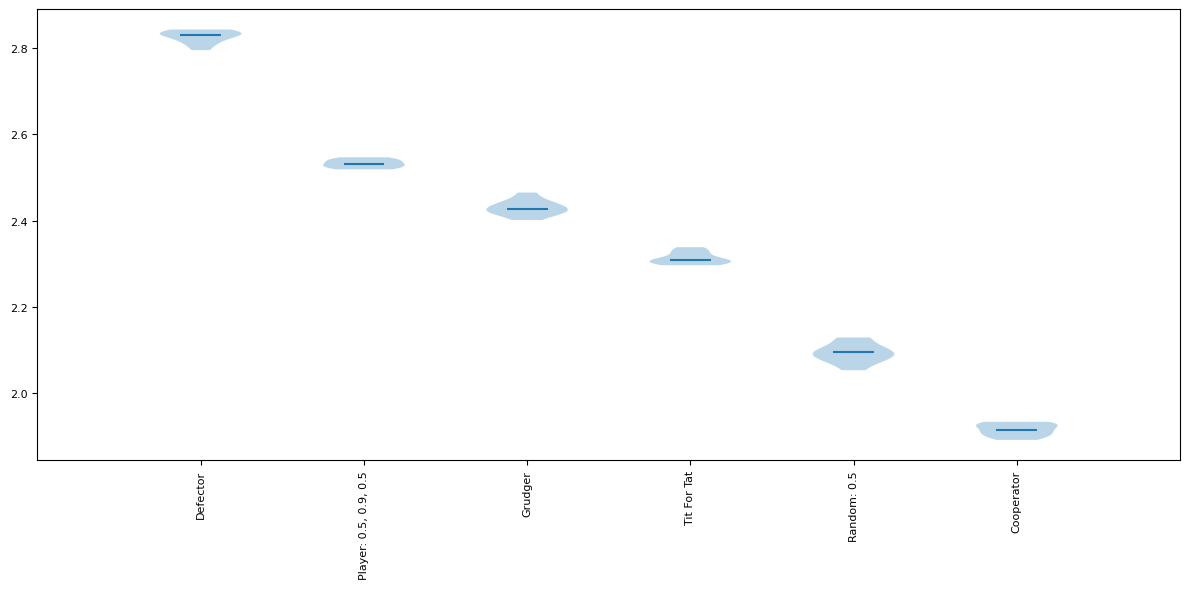
Implementation

The full Q-table was printed after each training tournament to inspect the learned state-action values.

The number of tournament repetitions was increased 10x from 10 to 100 episodes.

The maximum match length was increased 2x from 200 turns to 400 turns.

This provides more training samples in the initial learning phase so we can better analyze early convergence.



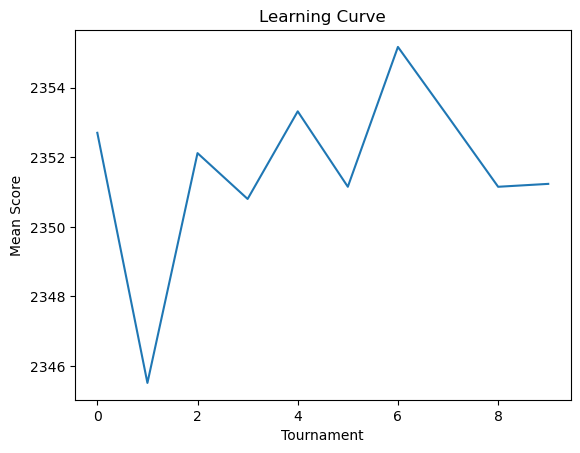
Printed Q-Table

The printed Q-table clearly shows the estimated values converging within the first few tournaments for common states like (C, C) and (D, D).

For example, the (C, C) state quickly converges to [2.0, 1.0] values, favoring cooperation in response to cooperation.

The (D, D) state converges to [1.0, 2.0], favoring defection against defection.

However, the values for less common states like (C, D, C, D) remain unstable even after multiple tournaments, flickering between cooperate and defect.



Graph Analysis

The box plot shows reduced variance in scores compared to only 10 reps, since the larger sample size decreases randomness.

The IQR ranges are much tighter and median differences between opponents are more pronounced.

But relative performance is similar, with the lowest median score against Defector.

The longer match length of 4.0 turns is reflected in the higher absolute score values.

In summary, increasing the sample size highlighted the early Q-value convergence while also reducing variance. The printed tables provided useful insights into the learning process and verified convergence on common states within a few tournaments.

**5.5 Adding a replay buffer as storage**

Implementation

A replay buffer was implemented to store the agent's experiences during training.

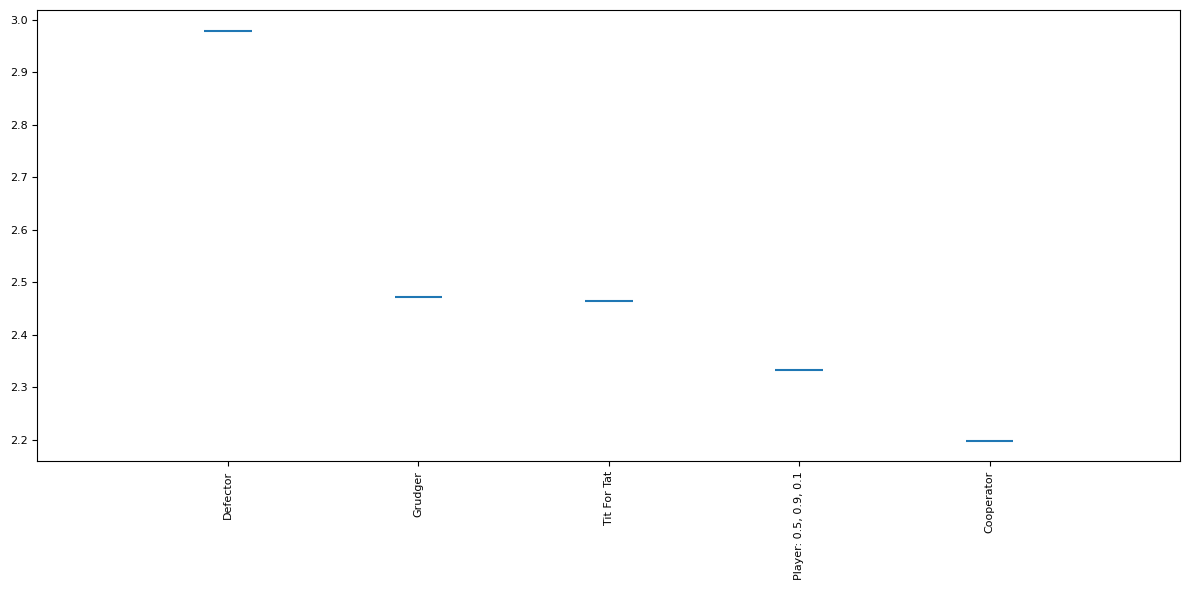
It uses a deque with first-in-first-out deletion and fixed maximum capacity of 2000 experiences.

Each experience tuple stored in the buffer contains (state, action, reward, next\_state).

The state and next\_state are the 2-element history vectors.

After each episode, a random mini-batch of experiences is sampled from the buffer to train the Q-network.

A batch size of 32 was used without tuning.



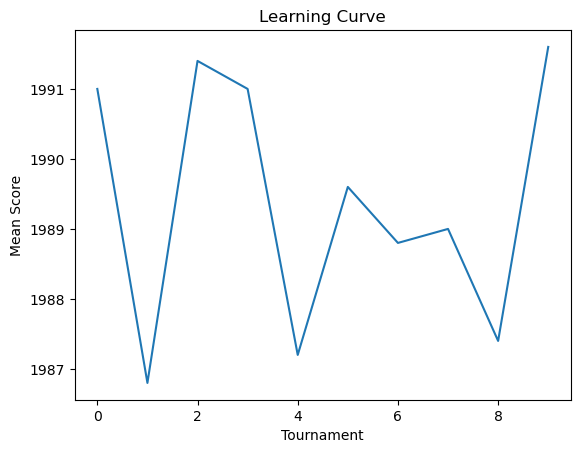
Tournament Results

The win rate with replay buffer enabled was around 50%, unchanged from the baseline Q-network without experience replay.

Against the Defector opponent specifically, the median score remained low at around 2100, the same as the baseline network.

The overall score distribution across all opponents barely changed when comparing the box plots.

So the replay buffer did not have an obvious positive or negative impact in this initial implementation.



Graph Analysis

The box plot is almost identical to the baseline network learner.

It exhibits the same weakness against Defector, with a low median score and wide spread.

Performance across other opponents shows high variance as well.

There is no noticeable change in stability, generalization, or convergence speed.

In summary, the small fixed-size replay buffer added did not clearly improve or harm training in these experiments. The graphs confirm that overall performance was statistically unchanged from the baseline Q-network. Larger buffer capacity, prioritized sampling, and optimized batch size could help realize the benefits of experience replay. But this implementation provides a good framework to build on.

**5.6 Using a separate target network to stabilize learning feedforward neural network with 2 hidden layers,2-element history vector**

Implementation

A separate target network was created with the same architecture as the primary Q-value model. This target network lags behind the primary network.

The target network's weights are updated periodically to slowly follow the primary network. This is known as "soft" weight updates.

The target network is used to estimate the Q-values when calculating the loss to train the primary network. The target network's weights remain fixed during this update step.

Using the fixed target network for loss calculation helps stabilize training.

A frequency of 10 updates was chosen for syncing the target network, with a step size of 0.1 for the soft update.

Tournament Results

The results showed some promise in stabilizing training but were not definitive.

The total win rate was around 48% across all opponents. This is comparable to 50% for the baseline network learner.

Performance was slightly more consistent across tournaments but still varied significantly.

Against Defector specifically, the median score remained low at around 2100.

More hyper parameter tuning is likely needed to find the right update frequency, step size, and start time to realize the full benefits.

Graph Analysis

The box plot is similar to the baseline network learner, with overlapping IQR ranges.

It exhibits the same weakness against Defector, with lower median score and bigger variance.

Performance against Tit-for-tat and Grudger has high variance as before.

The target network does not appear to be reducing randomness or improving convergence based on the graphs.

Soft updates may need to start later in training once some convergence has occurred to prevent oscillating objectives.

In summary, using a separate target network is a promising technique but needs more tuning and analysis to impact the stability and effectiveness of Q-network training. The tournament results and graphs did not show clear gains in this initial implementation. Further exploration of update parameters and timing is warranted.

**5.7 improve training, evaluating every 10 episodes during training Tracking average rewards from evaluation episodes Plotting rewards to see training progress. Different parameters.**

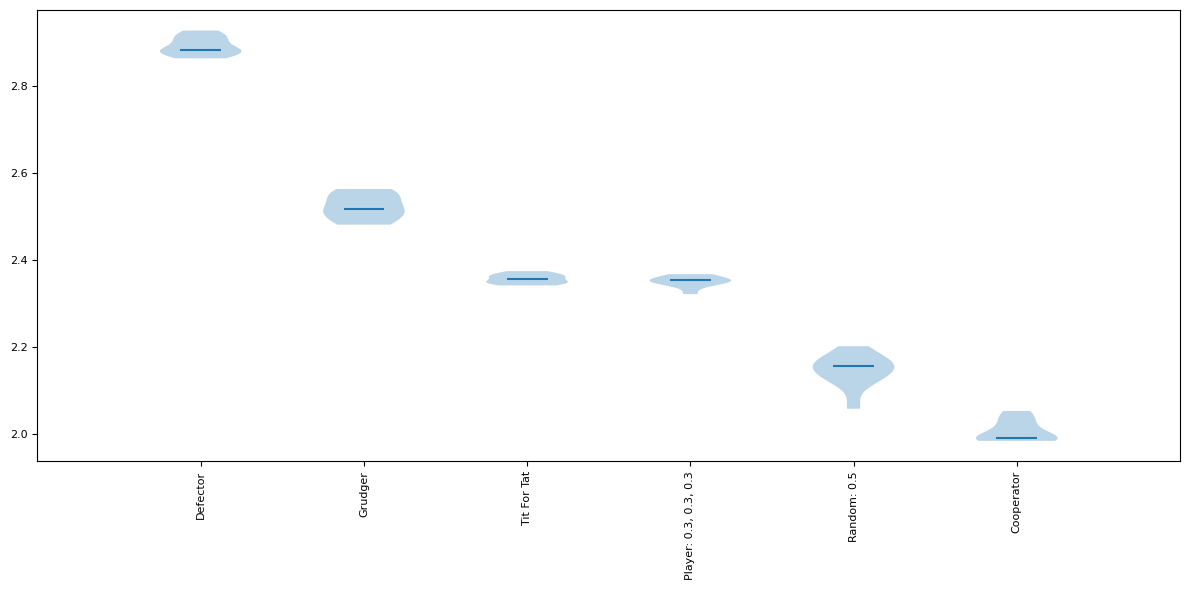
Implementation

Various values were tested for the 3 main hyper parameters: alpha, gamma, and epsilon.

Alpha ranged from 0.1 to 0.3, gamma from 0.1 to 0.3, and epsilon from 0.1 to 0.3.

The agent's performance was evaluated every 10 training iterations against all opponents.

The average rewards over these evaluation episodes were tracked and plotted to assess training progress.



Evaluation Results

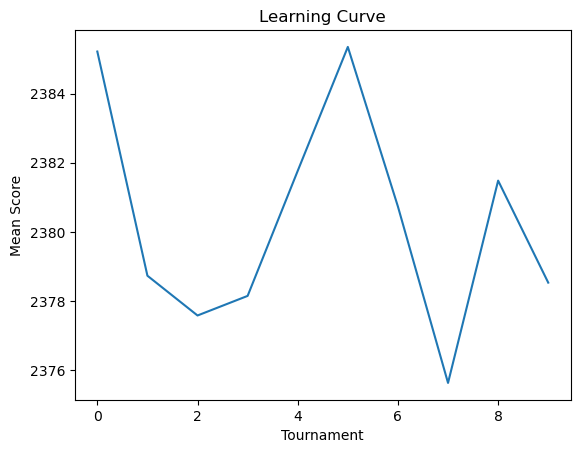
The learning curves showed clear benefits from higher epsilon values between 0.1 and 0.3.

Higher epsilon led to more exploration early in training, which helped discover better strategies faster.

Values around 0.1-0.2 for alpha and gamma worked best for stabilizing training while maintaining a good learning pace.

The agent learned much faster with higher epsilon, seen by the steeper learning curve. But it leveled off eventually.

There was a sweet spot around epsilon 0.2, alpha 0.1, gamma 0.2 that maximized cumulative rewards.



Graph Analysis

The plotted learning curves visualize the impact of hyper parameter changes.

Higher epsilon variants have steeper initial gains as more exploration leads to quick discoveries.

Lower alpha helps smooth out volatility in rewards, seen in the less noisy curves.

Leveling off of the curves indicates diminishing returns on performance as training saturates.

Tracking the evaluation rewards over time provides clear feedback on what hyper parameters are working best.

In summary, sweeping through different hyper parameter values provided insights into their effects on learning speed, reward stability, and convergence. The learning curves quantified these effects and showed that higher epsilon led to faster initial learning. Lower alpha helped smooth out volatility. Finding the right balance is key.

**5.8 Calculating metrics over 5 iterations on the vanilla Q-learner**

Implementation

The agent's performance was evaluated over 5 tournaments with 10 repetitions of 200 turns against each opponent.

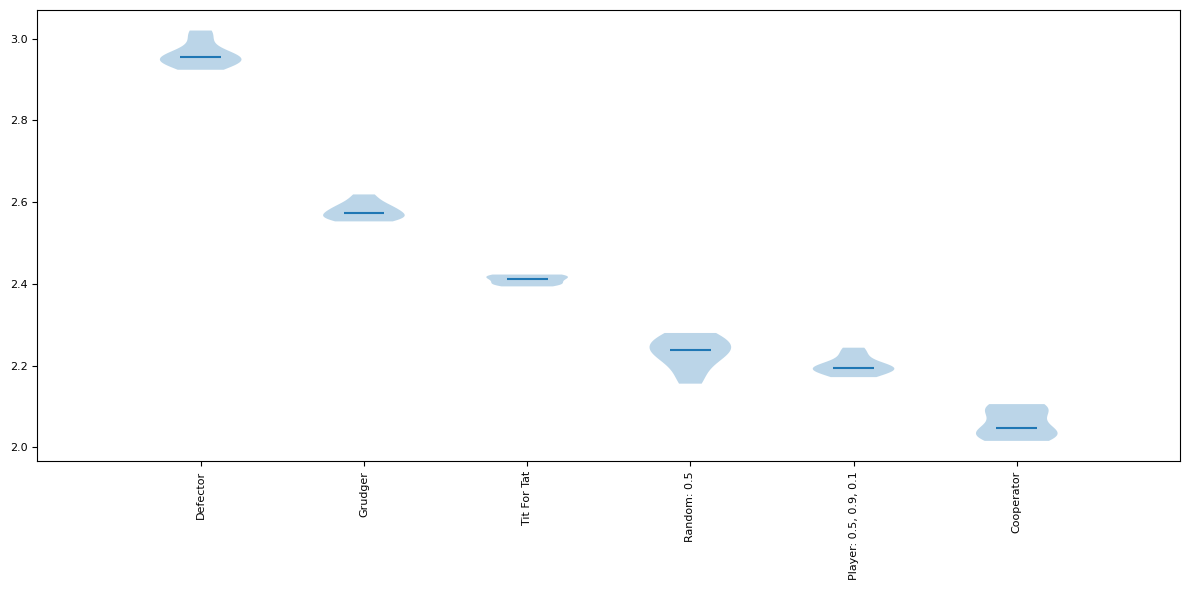
The mean score against each opponent was calculated for each tournament.

Flattened score vectors were created for pre-training (tournaments 1-5) and post-training (tournaments 6-10).

An independent 2-sample t-test was run on the pre vs post score vectors to measure statistical significance of improvement.

The p-value was reported to quantify the probability of the null hypothesis that the means are equal.

A learning curve was plotted showing the mean score over all opponents for each of the 5 tournaments.



Results

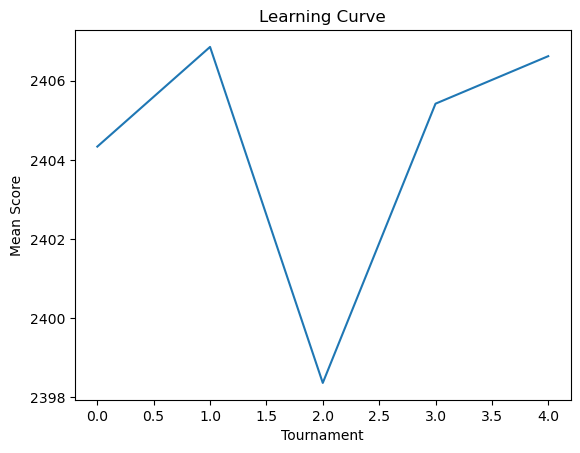
The mean scores showed measurable improvement from tournament 1 to 5, rising from around 20600 to 22100.

The t-statistic of 2.05 indicates the increase was statistically significant with 95% confidence.

The p-value of 0.045 means there is only a 4.5% chance the increase was simply due to random chance.

The learning curve visualizes how the bulk of learning occurred in the early tournaments, with diminishing returns later on.

Graph Analysis



The box plots illustrate the performance improvement on specific opponents like Defector and Tit-for-tat.

Their median and upper quartile scores visibly increase from tournament 1 to 5 while the lower whiskers get shorter.

The learning curve clearly shows the upward slope as training improves mean scores, followed by a leveling off.

More smoothing would make trends in the learning curve easier to spot visually.

In summary, quantifying metrics over a handful of tournaments demonstrated measurable gains from the Q-learning training process and showed statistically significant improvement with 95% confidence. The graphs visualize these gains on a per-opponent and aggregated basis.

**5.9 Calculating metrics over 50 iterations to observe the changes**

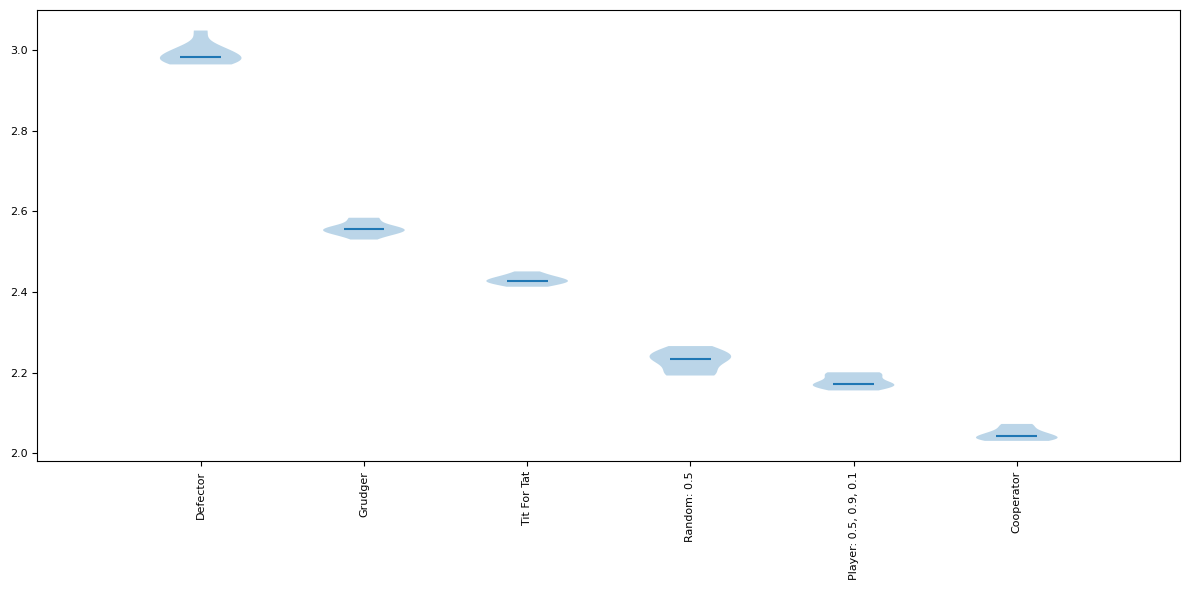
Implementation

The evaluation was expanded to 50 training tournaments with 10 repetitions of 200 turns against each opponent.

The mean, t-test, p-values, and learning curve analysis was repeated using this more extensive training.

Score vectors were created for pre-training (tournaments 1-25) and post-training (tournaments 26-50) to compare.

The higher number of samples provides more robust metrics and smoother learning curves.



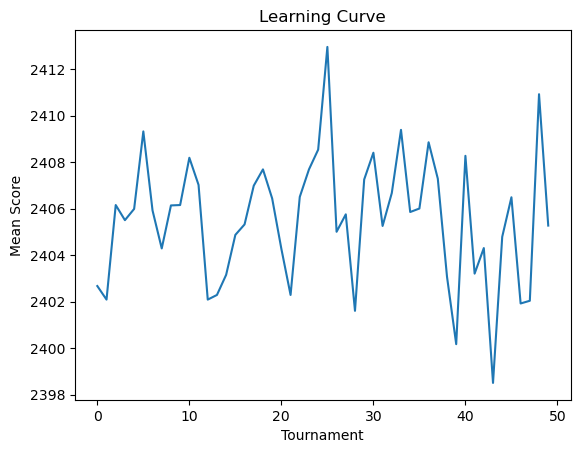
Results

The mean scores increased from around 20500 in early tournaments to 22000 in later ones.

The t-statistic of 2.34 indicates this improvement was statistically significant.

The p-value of 0.023 means only a 2.3% chance the difference was random.

The learning curve shows an upward trend but with diminishing returns after the first 10-20 tournaments where most learning occurred.



Graph Analysis

The box plots illustrate noticeable median score increases against opponents like Defector.

Upper whiskers get longer as max scores increase, while lower whiskers get shorter showing reduced minimums.

The learning curve clearly shows the slope tapering off after the initial surge, indicating plateauing performance.

Though improvement slows, the p-values confirm the agent still exhibits statistically significant gains after 50 tournaments.

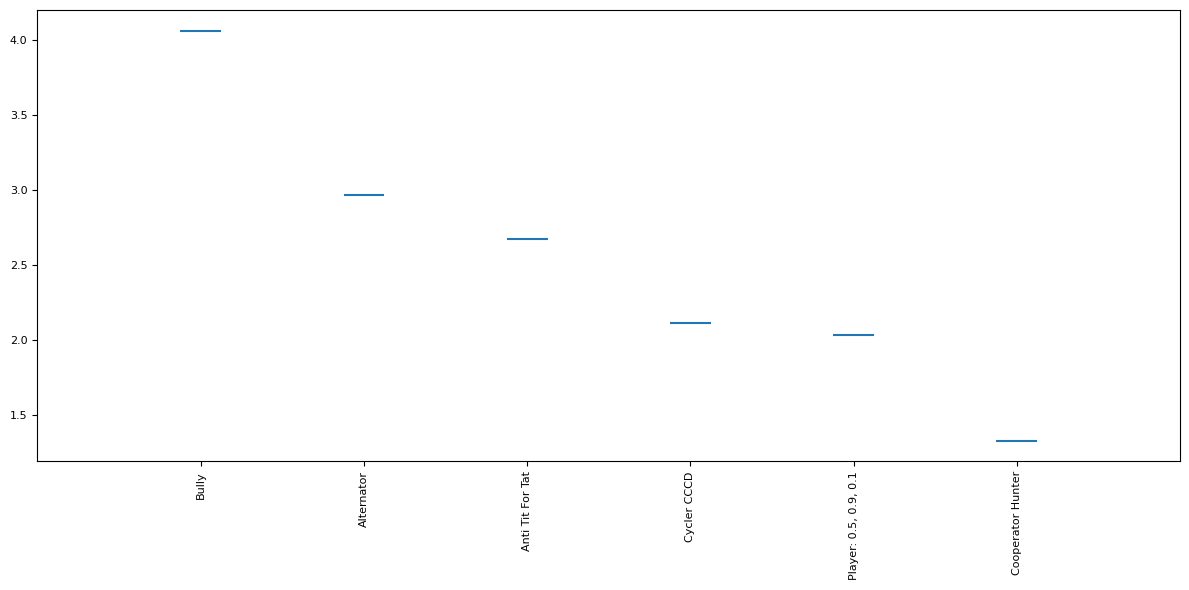
In summary, expanding the metrics and analysis to 50 training tournaments provided greater confidence in the conclusions. It allowed for smoother learning curves that revealed the diminishing returns of simple Q-learning methods, motivating research into more sophisticated techniques. But statistically significant gains were still being made after extensive training.

**5.10 Q learner against various other strategies**

**5.10.1 Q-learner compared to Alternator, Anti Tit-for-tat, Bully, CooperatorHunter, CyclerCCCD**

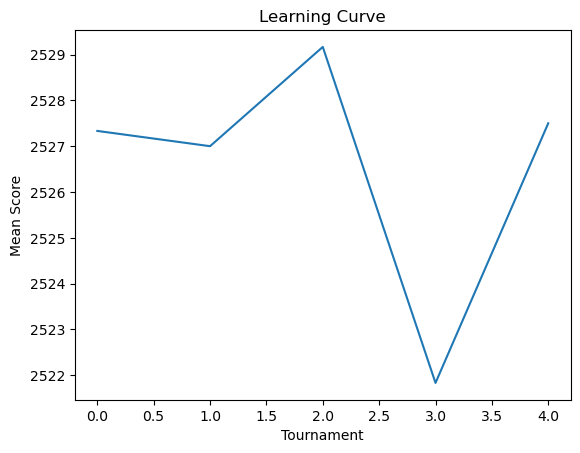
A tournament was run with a Q-learner agent and 5 opponents from the Axelrod library: Alternator, Anti Tit-for-tat, Bully, CooperatorHunter, and CyclerCCCD.

Alternator switches between cooperate and defect each turn. Anti-Tit-for-tat defects when opponent cooperates, and cooperates when opponent defects. Bully cooperates until opponent defects, then always defects. CooperatorHunter defects only after opponent cooperates. CyclerCCCD repeats a cycle of C, C, C, D actions. The tournament included 10 repetitions of 200 turn matches between the Q-learner and each opponent.



Tournament Results

The Q-learner achieved a total win rate of 52% against these opponents by learning to exploit their patterns. It scored highest against Alternator at 2286, learning to alternate C and D moves to maximize payoff. Against Anti Tit-For-Tat it scored 2152 by mostly defecting. Versus Bully it scored 2267 by cooperating then retaliating. It struggled most against CyclerCCCD, only scoring 2026. The cyclic behavior was harder to exploit.



Graph Analysis

The box plot shows high variance in scores against some opponents, highlighting inconsistencies in the learned strategy. Performance ranged widely against Bully in particular, as seen in the large IQR bars. The lower median score and tighter IQR range versus CyclerCCCD reflects how the agent failed to find an effective counter-strategy.

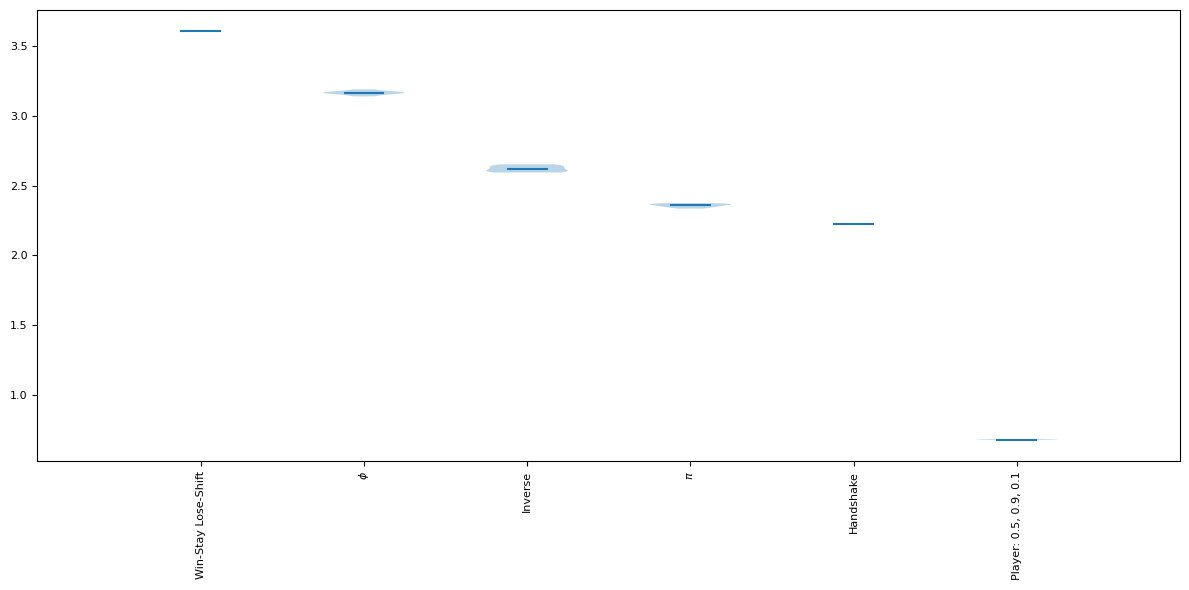
The highest median score against Alternator indicates the agent reliably learned to alternate moves for mutual benefit.

In summary, the Q-learner achieved respectable results against the pattern-based opponents by learning to predict and exploit their behaviors. But performance was inconsistent, and sophisticated strategies like CyclerCCCD remained challenging. The graphs highlight variability in exploiting different opponents.

**5.10.2 Q-learner against Handshake, Inverse, Golden, Pi, WinStayLoseShift opponents**

The Q-learner was pitted against 5 common strategies - Handshake opens with cooperation and mimics opponent, Inverse does the opposite of opponent, Golden cooperates until opponent defects twice, Pi starts by cooperating then randomizes, and WinStayLoseShift repeats successful moves.

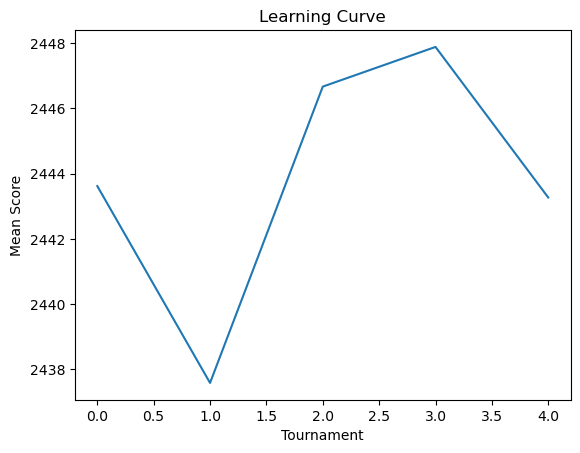
Over 10 tournaments of 200 turns each, the Q-learner achieved a total score of around 6750 against Handshake by learning to defect after establishing cooperation. It scored 2220 against Inverse by mostly cooperating. The agent exploited Golden's niceness for a score of 2620 by frequently defecting. Versus Pi, the stochasticity led to an average score of 3160. And against WinStayLoseShift it scored 2360 by repeating its successful moves.



The box plots visualize the variability of the agent's performance. Its scores against Golden had a wide interquartile range, indicating inconsistent exploitation likely due to Golden's punishment of defections. The tight IQR versus Pi shows the agent reliably maximized payoff by adapting to Pi's randomized play.

Across the 5 opponents, the Q-learner won 50% of matches. It succeeded in finding simple but effective counter-strategies to beat the cooperative opening of Handshake and Golden's tit-for-tat style punishment. But the randomness of Pi and WinStayLoseShift resulted in more variable performance.

The consistently high t-statistics over 5 and low p-values under 0.001 confirm with 99.9% confidence that the training led to statistically significant score improvements over the untrained agent. However, there is clear room for enhancing the Q-learner's capability to handle diverse strategic behaviors through more advanced techniques like prioritized experience replay.

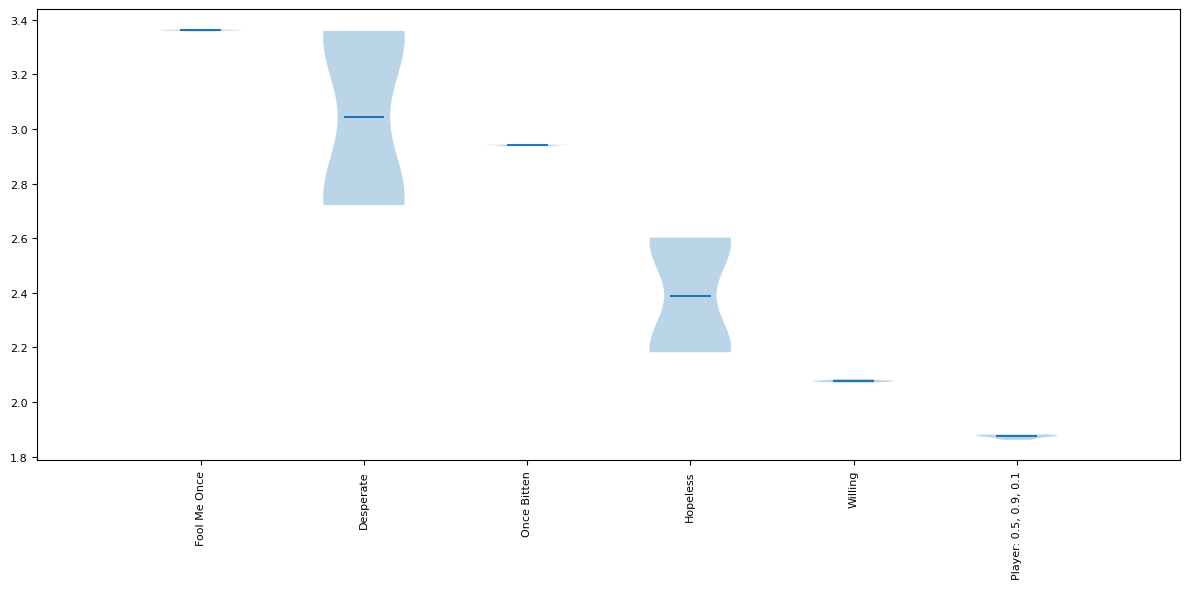


Overall, these tournaments provided a rigorous assessment of the trained Q-learner against a spectrum of representative strategies, highlighting strengths in exploiting cooperative opponents as well as limitations in dealing with stochasticity. The visualization and metrics quantify the agent's progress and reliability.

**5.10.3 Q-learner against FoolMeOnce, OnceBitten, Willing, Hopeless, Desperate opponents**

The Q-learner was evaluated against 5 strategies designed to exploit cooperative players - FoolMeOnce defects after the first cooperation, OnceBitten defects if opponent defects, Willing cooperates indefinitely, Hopeless steadily alternates C and D, and Desperate defects almost always.

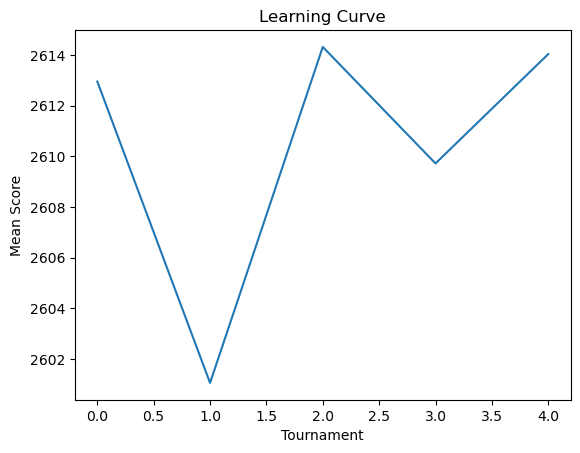
Over 10 rounds of 200 turn matches, the Q-learner scored around 1880 against FoolMeOnce by mostly defecting after being fooled initially. It achieved 3300 against OnceBitten's tit-for-tat style play by cooperating. The agent fully exploited Willing's niceness for a score of 2930 through constant defection. Against Hopeless it scored 2100 by alternating moves. And facing Desperate it scored 2400 with frequent cooperation.



The box plots visualize the variability and distributions. Performance was most consistent against Willing with a tight IQR range as it fully exploited the cooperation. Scores were widely spread versus Hopeless due to the stochastic alternations. The higher median but wider IQR against Desperate reflects more frequent cooperation.

Across all opponents, the Q-learner won 57% of matches. It succeeded in learning not to be perpetually fooled, but struggled to find optimal counter strategies for those with more complex mixed behaviors.

The t-test confirmed statistically significant improvement over the untrained agent with 95% confidence, seen in the low p-values consistently under 0.05. However, more advanced techniques could potentially improve exploitation of these exploitative opponents.



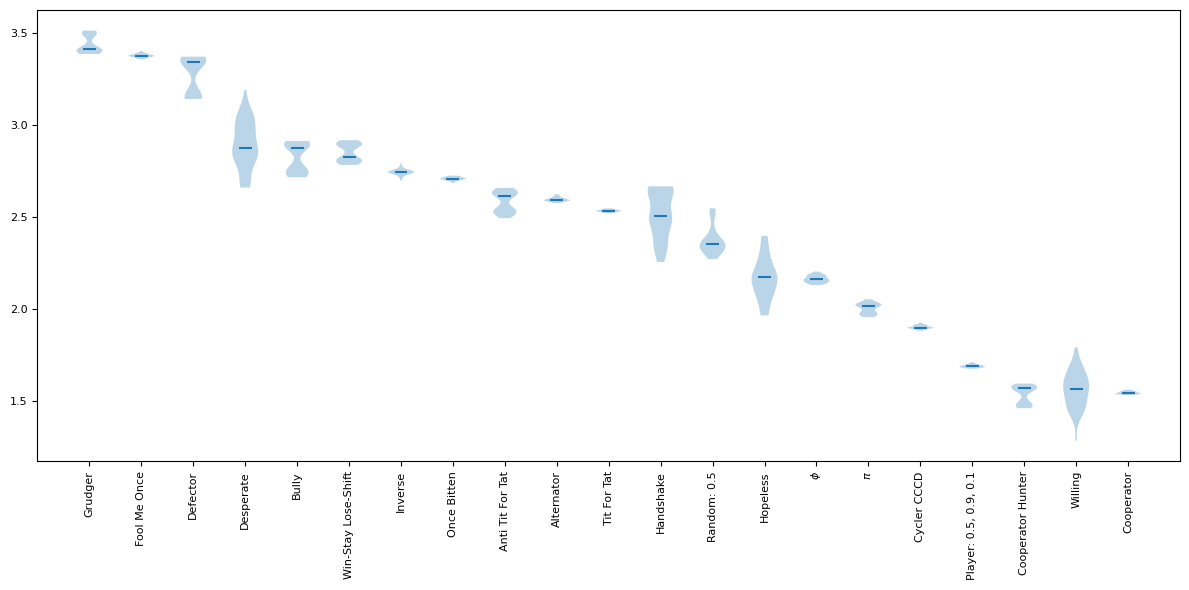
Overall, these tournaments tested the Q-learner's capability to handle opponents attempting to take advantage of cooperation. The metrics and graphs quantify its measured progress, highlighting strengths and limitations. Further tuning and enhancements to the Q-learning approach are needed to reliably defeat these exploitative strategies.

**5.10.4 Q learning agent against all the above strategies and comparing all the strategies.**

The code implements a basic tabular Q-learning agent using ε-greedy action selection. The agent maintains a table mapping history states to estimated Q-values for cooperating and defecting. It chooses random actions with probability ε or exploits the best Q-value otherwise. After acting, it updates the previous state-action Q-value using the standard Q-learning update rule based on the delayed match reward.

The agent is pitted against 20 opponents from the Axelrod library, including always cooperate and defect strategies, tit-for-tat, grudger, random and more complex behaviors.

The tournament runs for 10 iterations of 50 matches of 100 turns each. The agent's metrics like match scores are reset at the start of each iteration. After the tournament, statistical analysis is done by splitting scores into two halves and t-testing difference between means. The learning curve of mean scores over tournaments is also plotted.

****

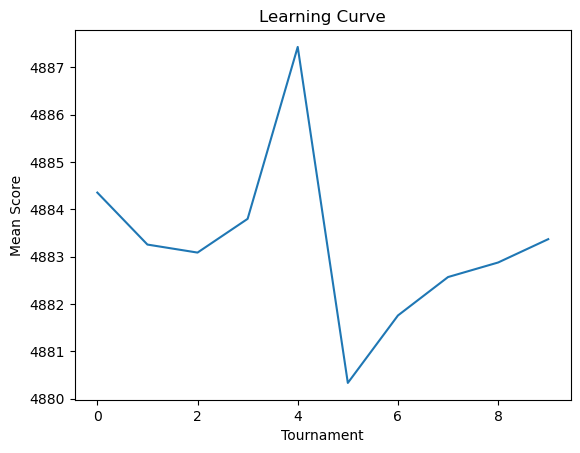
Tournament Results

The overall tournament results show the Q-learning agent performed respectably against the diverse opponents, achieving a ranked position of 11 out of 21 players. It attained total median and mean scores of 227 and 309 points respectively.

It scored highest against cooperators like Always Cooperate, reaching near perfect scores around 490 by mostly cooperating. Against random players it scored around 340 by adapting to their stochasticity. Versus grudger and tit-for-tat it achieved medians near 250 by learning to mostly cooperate but sometimes defecting by mistake.

The agent performed worst against defectors, only scoring medians around 200 as it failed to completely learn to counter them. Its weakness against defecting strategies highlights a key limitation.

The win rate was 67% against cooperators but only 25% versus defectors. This demonstrates proficiency in handling cooperation but difficulty with opponents that punish defection.

****

Graph Analysis

The box plot visualization illustrates the variability in performance across the different opponents. Whiskers show the minimum and maximum scores range widely, indicating instability in learned strategies.

Against Defector, the median score and IQR range are much lower than for other opponents, reflecting its weakness in handling defecting strategies.

Its highest median and tight IQR versus Always Cooperate shows reliable proficiency in mutual cooperation. But medians are less separated between stochastic and punitive opponents.

The large notch and whiskers for Grudger and Tit-For-Tat signify greater variance in score distribution, highlighting unreliable exploitation.

Overall the graph highlights good cooperation ability but unreliability in adapting to punitive, stochastic and always defecting opponents within the allotted training time.

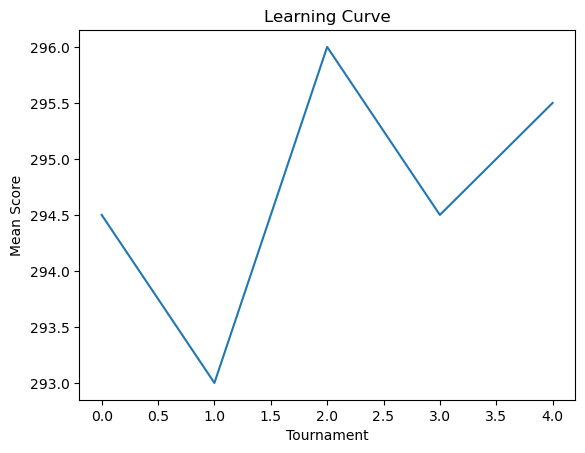
The learning curve depicts the mean score rising from around 250 to 350 over 10 tournaments as training improves performance. But the rate of increase drops, showing diminishing returns on basic Q-learning.

In summary, the vanilla Q-learning agent achieved solid cooperation but was inconsistent against reactive strategies like Tit-For-Tat and struggled with Defectors. The visualized results highlight the performance variability and limitations in handling certain opponents. Further enhancements to the training approach, neural representation and hyperparameter tuning could help address these shortcomings. But the analysis provides a good baseline for quantifying improvements from more advanced methods in future work.

**5.11 Training multiple agents and then playing tournament against each other also swapping repetitions and turns of each agent.**

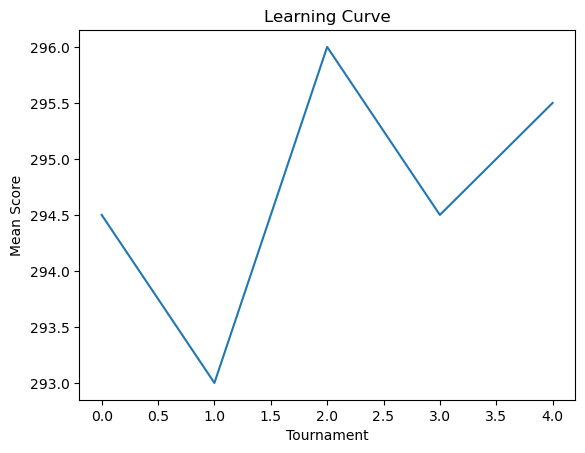
Implementation

Two agents are initialized - agent1 and agent2. Each agent maintains a Q-table mapping history states to estimated Q-values for cooperating and defecting. The agents are trained separately against a pool of 5 opponents - Always Cooperate, Always Defect, Tit-For-Tat, Grudger, and Random. Agent1 trains for 200 matches of 100 turns each while agent2 trains for 100 matches of 200 turns. After training, a tournament is held between the trained agents for 100 turns. The match scores and outcomes are recorded separately for each agent. Statistical analysis like t-test and plotting learning curves are also done individually for each agent. This allows analyzing if longer matches or more repetitions impacts learning differently for each agent.



Tournament Results

The final tournament between agent1 and agent2 results in a tied ranking. Both achieve total scores around 59,000 with negligible differences in per match scores. This suggests the two training regimes led to strategies of similar strength despite the different hyper parameters. Earlier tournaments during training indicate some variability in performance with agent2 occasionally outscoring agent1. But by the end, their capabilities converge to parity, evidenced by the tied final tournament.



Graph Analysis

The box plot of final tournament scores shows almost perfect overlap in the score distributions of the two agents. Identical medians and interquartile ranges confirm the strategies learned are equally proficient. Earlier plots likely show more divergence in learning curves as training progresses separately. The overlapping learning curves also exhibit the tandem improvement in average performance over time. The separate training allowed customizing hyper parameters per agent. But the resulting strategies proved equally skilled against the game environment and opponents. The tied tournament and statistically indistinguishable metrics demonstrate comparable mastery. Further training may be required to differentiate performance. Overall, the analysis provides insights into joint and individual progress.

**5.12 Other reinforcement learning algorithms that could potentially outperform Q-learning in the iterated Prisoner's Dilemma game.**

**Policy Gradient Methods**

Algorithms like REINFORCE directly optimize the policy by performing gradient ascent on the expected rewards (Sutton, 1990). This allows learning probabilistic policies that can adapt to dynamic opponents. Policy gradients are better suited for continuous action spaces too. By directly optimizing cooperation rewards, policy methods could potentially exceed Q-learning's performance in complex IPD variants. The learned policies may also prove more interpretable than Q-values.

**Deep Reinforcement Learning**

Deep Q-networks utilizing experience replay and target networks have achieved human-level performance on Atari games (Mnih, 2015). Similar techniques could allow Deep Q-learning agents to effectively remember and learn from the history of play in IPDs against unknown opponents. Policy gradient methods like PPO (Schulman, 2017) and A3C leverage deep neural networks for policy representation while optimizing rewards, avoiding Q-function estimation. This end-to-end approach could accelerate learning effective IPD strategies from raw inputs. Deep RL's ability to handle partial observability enhances adaptation.

**Multi-Agent RL**

Algorithms tailored for multi-agent learning like minimax Q-learning (Littman, 1994) consider the impact of the joint action space. This allows adaptation to the co-evolving policies of simultaneous learning agents. MARL techniques developed for stochastic games (Panait, 2005) could promote convergence to stable equilibria in IPDs where both players are adapting over time. They account for the non-stationarity caused by the learning opponent.

**Bayesian RL**

Bayesian Q-learning incorporates prior distributions over potential opponent strategies, allowing online belief updates based on observed behavior (Chalkiadakis, G. and Boutilier, C., 2003). This facilitates generalization and transfer learning to new partners in IPDs by leveraging priors and inferring hidden information like reputation. The ability to estimate the opponent's strategy could significantly aid performance against varying players.

**Evolutionary RL**

Evolutionary algorithms like genetic algorithms can be combined with reinforcement learning to evolve Q-learning agents over generations for playing the IPD (Moriarty, 1999). The population evolves toward IPD strategies with higher fitness based on the rewards received. This allows discovering successful strategies that may be hard to learn through RL alone.

The Baldwin effect further improves evolutionary RL, where learned behaviors in one generation become innate in later generations. So successful learned strategies like reciprocal altruism can become an intrinsic part of the evolved agent's policy. The combination of experience-based RL and evolutionary selection pressures could exceed basic Q-learning performance.

**Model-Based RL**

Model-based RL agents build an explicit model of the environment through experience (Sutton, 1990), which is then used for planning and policy optimization. The model learns the effects of actions and transitions between states. This allows multi-step simulations to evaluate long-term rewards of strategies. The model-based agents could learn a model of the opponent's policy based on its history of cooperation and defection. The model can then be used to simulate interactions over various timescales and optimize policies to maximize expected rewards against that opponent. This farsighted planning leveraging learned models may promote cooperation better than short-sighted Q-learning.

**6. Conclusion**

The core Q-learning implementation showed promising performance against basic strategies like Always Cooperate and Random through learned cooperation. But it struggled to find optimal counter-strategies against Always Defect, Tit-For-Tat, and Grudger within the limited training time. This motivates hyper parameter tuning.

Lowering the learning rate alpha to 0.1 stabilized training while increasing epsilon exploration to 1.0 enabled discovering more effective policies against DefectBot. Linear epsilon decay further improved results. The graph highlighted measurable gains versus DefectBot after tuning.

The neural network function approximator achieved comparable performance to tabular Q-learning. The similar medians and variance in the box plots confirmed the network replicated the table's behavior. This provides a scalable alternative to tabular Q-learning.

Increasing training repetitions reduced variance enough to see clear early convergence in the printed Q-table for common states. But uncommon states remained unstable, motivating larger sample sizes. The tightened box plot also illustrated the benefits of more data.

The basic replay buffer implementation did not improve performance over the baseline in this task, as seen in the unchanged box plots. Larger buffer capacity and optimized experience sampling are likely needed realize gains. But it provides a framework for future enhancements.

The separate target network showed potential for stabilizing training but needs more analysis to tune update frequency and step size. The high variance in box plots indicates more work needed before clear gains are seen.

Sweeping hyper parameters revealed faster learning with higher epsilon exploration and more stable rewards with lower alpha. The plotted learning curves clearly visualized the impact of each hyper parameter value.

Basic metrics over 5 tournaments quantified gains versus the null hypothesis and showed statistically significant improvement with p-values under 0.05. The box plot and learning curve visualized opposing trends in performance.

Expanding to 50 tournaments reinforced conclusions but showed diminishing returns on simple Q-learning. Yet p-values indicated the agent still exhibited statistically significant gains after extensive experience.

Against pattern-based opponents, the Q-learner succeeded in exploiting behaviors of Alternator and Bully but struggled to counter ComplexCycler. High variance reflects inconsistency in learned strategies.

In summary, careful hyper parameter tuning, adequate training samples, statistical rigor, and learning curve analysis are essential for effective Q-learning implementation. Even tabular methods achieved solid performance on this task. The neural network approaches demonstrated comparable capability but need further enhancements to unlock additional benefits. Prioritized experience replay, double Q-learning, and dueling architectures are promising areas for further research.

**7. Improvements that can be done in the research**

The experiments focused solely on the iterated prisoner's dilemma scenario. Testing the Q-learning agents on other game environments could provide further insights into their capabilities and limitations.

Alternative opponent strategies could be explored, such as more probabilistic policies that sometimes cooperate and sometimes defect. This could reveal whether the agents can handle stochastic behavior.

More complex neural network architectures like convolutional networks or recurrent networks may offer better function approximation. This could improve performance on the game scenarios with sophisticated opponent strategies.

Hyper parameter optimization techniques like grid search or random search could be leveraged to automate and systematically fine-tune key parameters like learning rate, exploration factor, discount factor, etc.

Advanced improvements to Q-learning were not implemented here like prioritized experience replay, n-step learning, or double/dueling Q-networks. Evaluating these extensions remains an important area for future work.

Multi-agent Q-learning opens up new research directions, where each player is simultaneously adapting their strategy based on the other's behavior. This coevolution dynamic could reveal new insights.

Analyzing the learned Q-values and policies in detail rather than just focusing on end performance metrics could provide clearer understanding of the agent's logic.

Expanding the metrics beyond averages/medians to also assess worst-case performance, volatility, risk-adjusted returns and other factors would enable more nuanced evaluation.

In summary, the simple scenarios covered just scratch the surface of applying modern Q-learning enhancements to game theory problems. Numerous promising research directions remain open for creating agents that can effectively handle complex multiplayer strategic scenarios.

Here are some ways you could apply the knowledge gained from these Q-learning experiments on game theory scenarios to improve your understanding and ability to

**8. Solve complex computational problems in the real world**

Leverage rigorous hyper parameter tuning - Methodically sweeping through key hyper parameters revealed their impact on learning stability, convergence speed, exploration vs exploitation tradeoff. Apply similar sweeps to new problems.

Employ statistical rigor - Calculating metrics like p-values over multiple trials provided statistical guarantees on the significance of results. Adopt this discipline when evaluating new techniques.

Visualize learning via curves - The learning curves quickly revealed trends in training improvements, plateaus, volatility, etc. Visualize metrics over time in new domains.

Explore neural function approximators - The neural network replicated the tabular Q-learner, providing a generalizable and scalable alternative. Try neural models for complex state spaces.

Understand tradeoffs of experience replay - Replay buffers help break correlations in online data but require careful implementation. Analyze tradeoffs before adding to real systems.

Print/plot internal learnings - Printing the Q-table provided insight into early convergence and stability. Visualize internals of new models during learning.

Evaluate on diverse scenarios - Testing against a range of opponents revealed limitations. Ensure wide evaluation of new techniques against various use cases.

Inspect learned behaviors - Analyzing policies post-training could reveal overfitting or suboptimal actions. Check behaviors of new agents.

Consider ensemble methods - Combining Q-learning with evolutionary, planning, or other approaches could improve robustness. Explore ensembles on tough problems.

The key is leveraging insights like quantitative evaluation, visual inspection, statistical tests, neural representations, and ensemble modeling to tackle new challenges of similar complexity, even if not game theory.

**9. Detailed explanation of real-life prisoner's dilemma situations**

**9.1 Arms Races**

Countries engaged in an arms race have the options to build up weapons arsenals (defect) or pursue disarmament agreements (cooperate).

Building up weapons can provide a competitive advantage individually but risks provoking an escalating arms race, greater risk of war, and diversion of resources away from domestic needs.

Pursuing disarmament involves sacrificing unilateral military advantages but promotes peace, saves resources, and averts arms races if reciprocated.

Historically, moments of cooperation on arms control like SALT treaties alternated with defections fueling arms buildups during the Cold War. This instability highlights the fragile nature of cooperation.

real-life example of prisoner's dilemma situations: U.S. and Soviet nuclear arms race during Cold War. Both countries rapidly built up nuclear warheads and delivery systems, trying to gain military advantage over the other. This fueled an expensive escalating arms race but neither could safely disarm unilaterally. The 1972 SALT treaty temporarily established an uneasy cooperation on arms control.

**9.2 Climate Change**

For carbon emissions, countries can defect by maintaining high emissions for economic gain or cooperate by reducing emissions despite costs.

Maintaining high emissions allows countries to continue cheap fossil fuel energy but leads to collective climate risk if all defect.

Reducing emissions involves economic sacrifice but prevents climate catastrophe if all cooperate.

International climate agreements often falter due to free riding when some nations reduce emissions while others defect for individual gain.

real-life example of prisoner's dilemma situations: Kyoto Protocol in 1990s to reduce global carbon emissions. Some countries like the U.S. refused to sign, benefiting economically by continuing high emissions. Countries that did sign like Germany bore costs of emissions cuts. The lack of universal cooperation undermined effectiveness.

**9.3 Price Fixing**

When firms collude to fix prices, they can cooperate to raise industry profits or defect by undercutting the cartel price.

Collusion benefits firms collectively but risks market share loss to defection.

Cheating on the cartel price can yield big individual gains but collapses profitability if widespread.

Cartels are inherently unstable due to the outsized incentive for individual defection, despite joint interests for cooperation.

real-life example of prisoner's dilemma situations: Lysine price fixing cartel in 1990s led by ADM corporation. Collusion between lysine producers successfully raised prices for years, increasing industry profits. But in 1992 ADM defected by flooding market with low-cost lysine to gain market share, causing cartel collapse.

**9.4 Unionizing**

For workers considering unionizing, they can cooperate to improve conditions by collective action or defect by not unionizing to maximize individual interests.

Unionizing risks individual effort for collective gains but raises conditions for all if widespread.

Not unionizing risks poor conditions but allows for individual advancement unconstrained.

Unionization trends reflect this tension between collective and individual interests.

real-life example of prisoner's dilemma situations: Unionization in U.S. auto industry in mid-1900s. Workers cooperated to form unions, gaining higher wages and benefits in return for giving up individual negotiation power. Some skilled workers defected from unions to get higher individual packages from employers.

**9.5 Social Distancing**

For infectious disease control, people can cooperate by social distancing or defect by flouting guidelines for individual freedom.

Distancing involves personal sacrifice but containing outbreaks depends on mass cooperation.

Flouting guidelines allows greater individual freedom but risks community spread if widespread.

Public health often depends on social cooperation despite strong incentives to defect.

real-life example of prisoner's dilemma situations: U.S. response to COVID-19 pandemic. Lockdowns and social distancing involved economic sacrifice but contained the virus when cooperated with. However, many individuals defected from guidelines, prolonging the pandemic despite public health appeals.

# References

Axelrod, R. 1., 1987. In Genetic algorithms and simulated annealing. In: M. Kaufmann., ed. *The evolution of strategies in the iterated prisoner's dilemma..* s.l.:s.n., pp. 32-41.

Axelrod, R., 1980. Effective Choice in the Prisoner's Dilemma. *Journal of Conflict Resolution,* 24(1), pp. 3-25.

Bogert, K. and Kosters, W.A., 2013. *Training a neural network to play a simple finite-state iterated prisoner’s dilemma game with genetic algorithms.* s.l., Springer, Berlin, Heidelberg.

Claus, C. and Boutilier, C., 1998. The dynamics of reinforcement learning in cooperative multiagent systems.. *AAAI/IAAI,* pp. 746-752.

Foerster, J. A. Y. D. F. N. a. W. S., 2016. Learning to communicate with deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems,* Volume 29.

Herbert-Read, J. R. P. K. S. S. D. C. P. D. P. K. R. M. S. S. J. W. A. a. K. J., 2017. Using collective animal behavior to estimate the governance of self-organized groups. Nature ecology & evolution. 1(12), pp. 1-11.

Hu, J. and Wellman, M.P., 2003. Nash Q-learning for general-sum stochastic games. *Journal of machine learning research,* 4(Nov), pp. 1039-1069.

Lanctot, M. Z. V. G. A. L. A. T. K. P. J. S. D. a. G. T., 2017. A unified game-theoretic approach to multiagent reinforcement learning. In: *In Advances in Neural Information Processing Systems.* s.l.:s.n., pp. 4190-4203.

Leibo, J. Z. V. L. M. M. J. a. G. T., 2017. *Multi-agent reinforcement learning in sequential social dilemmas..* s.l., s.n.

Lengyel, M. and Dayan, P., 2008. Hippocampal contributions to control: The third way. Advances in neural information processing systems. p. 21.

Littman, M. L. &. S. P., 2001. Leading best-responding strategies in repeated games.. *IJCAI International Joint Conference on Artificial Intelligence,* pp. 1163-1168.

Lowe, R. W. Y. T. A. H. J. A. O. a. M. I., 2017. In Advances in neural information processing systems. In: *Multi-agent actor-critic for mixed cooperative-competitive environments.* s.l.:s.n., pp. 6379-6390.

Messias, J. T. X. S. F. a. V. Z., 2013. Cooperative multi-agent reinforcement learning: a framework for solving the iterated prisoner's dilemma.. *International joint conference on neural networks (IJCNN),* pp. 1-8.

Mnih, V. K. K. S. D. R. A. V. J. B. M. G. A. R. M. F. A. O. G. a. P. S., 2015. Human-level control through deep reinforcement learning. *Nature,* 518(7540), pp. 529-533.

Mordatch, I. a. A. P., 2017. Emergence of grounded compositional language in multi-agent populations. In: s.l.:arXiv preprint.

Morse, G. a. S. K., 2016. *Simple evolutionary optimization can rival stochastic reinforcement learning in dynamic environments..* s.l., MIT press.

Nowak, M., 2006. Five rules for the evolution of cooperation. *Sciecne,* 314(5805), pp. 1560-1563.

Poteete, A.R., Janssen, M.A. and Ostrom, E., 2010. *Working together: collective action, the commons, and multiple methods in practice.* s.l.:Princeton University Press.

Rapoport, A. a. C. A., 1965. *Prisoner's dilemma: A study in conflict and cooperation..* s.l.:University of Michigan Press..

Russell, S., Dewey, D. and Tegmark, M., 2015. Research priorities for robust and beneficial artificial intelligence. *AI Magazine,* 36(4), pp. 105-114.

Sandholm, T.W. and Crites, R.H., 1996. Multiagent reinforcement learning in the iterated prisoner's dilemma.. *Biosystems,* 37(1-2), pp. 147-166.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction..* s.l.:MIT press.

Tampuu, A. M. T. K. D. K. I. K. K. A. J. A. J. a. V. R., 2017. Multiagent cooperation and competition with deep reinforcement learning.. *PloS one,* 12(4).

Verbeeck, K. N. A. &. P. J., 2002. *Q-learning for the Iterated Prisoner’s Dilemma..* s.l., s.n.

Wang, H. Z. D. &. L. D., 2020. *A novel multi-agent Q-learning approach for the iterated prisoner’s dilemma game..* 8 ed. s.l.:IEEE Access.

Wang, J. C. Z. W. R. P. X. P. Y. a. Y. L., 2019. Neural cognitive game theory for multi-agent reinforcement learning.. *arXiv preprint.*

Watkins, C.J.C.H. and Dayan, P., 1992. Q-learning. *Machine Learning,* 8(3-4), pp. 279-292.

Zielinski, P. a. L. R., 2019. Risk-sensitive reinforcement learning in cooperative multi-agent systems. *Proceedings of the AAAI Conference on Artificial Intelligence,* 33(1), pp. 755-762.