Reinforcement Learning for Tichu

A Comparison of Deep Q-Networks and Proximal Policy Optimization

Bachelor Thesis

Faculty of Science, University of Bern

submitted by

Nicolas Wyss

from Wynigen, Switzerland

Supervision:

PD Dr. Kaspar Riesen

Institute of Computer Science (INF)

University of Bern, Switzerland

Abstract

Reinforcement learning (RL) has emerged as a powerful approach for developing autonomous agents capable of making complex decisions in uncertain environments. This thesis investigates the development and evaluation of RL agents for the card game Tichu, a four-player game characterized by imperfect information. While previous research has explored Tichu agents, this study aims to provide a comprehensive overview of two different RL methodologies to determine their effectiveness in this challenging game. In particular, we examine two distinct RL methodologies: the value-based Deep Q-Networks (DQN) and the policy-based Proximal Policy Optimization (PPO). Moreover, we integrate Behavioral Cloning (BC) as a pretraining technique to enhance agent performance by learning from expert data. Our evaluation includes testing these agents against each other and against two baseline agents. The results demonstrate that PPO, especially when combined with BC, outperforms DQN in the context of Tichu, achieving performance equivalent to that of an amateur human player. These findings suggest that PPO is better suited for the complex card game Tichu compared to DQN. Furthermore, incorporating BC significantly enhances agent performance, highlighting its effectiveness in improving strategic capabilities.

Acknowledgements

I am grateful for the guidance and support provided by my supervisor, PD Dr. Kaspar Riesen, throughout the development of this thesis. I also extend my thanks to the University of Bern for granting access to computational resources on UBELIX (https://www.id.unibe.ch/hpc), the HPC cluster at the University of Bern, where the calculations for this research were perfomed.

Contents

Chapter 1

Introduction

This thesis is situated within the domain of artificial intelligence (AI). According to John McCarthy [1], the goal of AI is to develop machines that behave as though they were intelligent. Designing AI systems that are both responsive and capable of effective learning has been an ongoing challenge, ranging from the development of robots that can perceive and react to their physical environments to the creation of software-based agents that engage with natural language and multimedia [2]. AI encompasses various subfields, including machine learning, pattern recognition, and computer vision each contributing to the advancement of intelligent systems.

Machine learning (ML) is a research area focused on enabling computers to learn and make decisions without being directly programmed [3]. ML is typically divided into supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, models are trained on labeled datasets, while unsupervised learning identifies patterns in unlabeled data. Reinforcement learning (RL) represents a different paradigm where models learn to make decisions through feedback received from interactions with the environment, focusing on maximizing rewards or minimizing penalties rather than using explicit labels [4].

RL techniques are typically classified into two principal categories: model-based and model-free approaches. Model-based methods rely on an explicit model of the environment and the agent. This model outlines the consequences of actions and the associated rewards, thereby enabling the inference of optimal policies. Conversely, model-free methods do not employ any explicit knowledge of the environment's dynamics or action outcomes. Instead, they learn the value of actions through trial and error. While model-based techniques can be computationally intensive, model-free techniques typically require a substantial amount of experiential data to learn effectively [5].

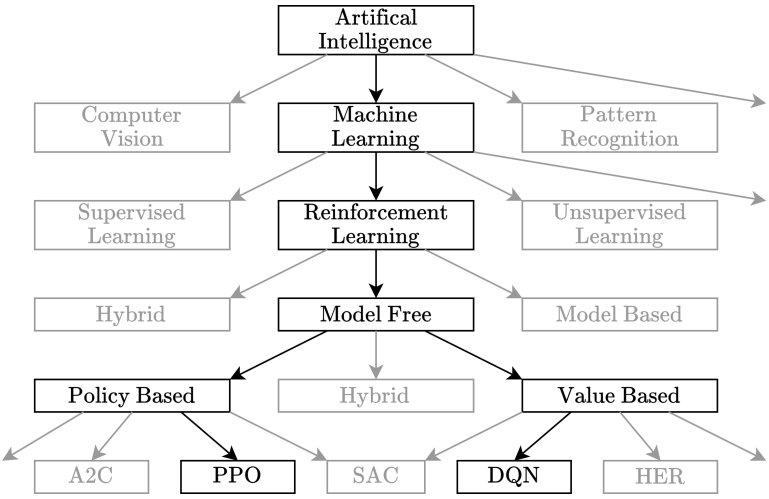


Figure 1.1: Hierarchical structure of AI, highligthing subfields used in this thesis

Moreover, model-free RL methods can be classified into value-based and policy-based approaches. Value-based RL methods concentrate on estimating the state-action value function, which is also referred to as the optimal Q-function, to guide decision-making. A notable example of a value-based algorithm is Q-Learning [6], in which the agent learns to approximate the Q-value function. Deep Q-Networks (DQN) build upon Q-Learning by employing neural networks to approximate the Q-function, thereby enhancing the algorithm's performance [7].

In contrast, policy-based RL methods seek to directly search through the policy space by employing parameterized function approximators such as neural networks. The policy gradient method represents a classic approach within this category, whereby policy parameters are updated based on the gradient of the expected long-term reward. An advancement in policy-based methods is the Proximal Policy Optimization (PPO) algorithm [8], which improves upon traditional policy gradients by incorporating clipping mechanisms and a dynamic learning rate to stabilize and enhance the learning process [9].

RL has diverse applications, including games, finance, healthcare, robotics, and transportation [10]. The intersection of ML and computer games dates back to the early days of AI, notably with Arthur Samuel's checkers-playing program [3]. Recent advancements have enabled RL algorithms to surpass human performance in increasingly complex games, underscoring the potential of this approach. The complexity of games can be classified based on several factors, which will be discussed in the following paragraphs.

A principal factor influencing the complexity of a game is the distinction between games with perfect and imperfect information. In perfect information games, such as chess or Go, the entire game state is observable to all players [11]. In 2016, the RL agent AlphaGo, which had learned by playing thousands of matches against other players, defeated the world champion in Go - the game regarded as the most challenging perfect information game for AI [12]. In 2017, AlphaZero was introduced, learning solely through self-play and surpassing AlphaGo's performance, achieving superhuman levels in Go, chess, and shogi [13].

In contrast, imperfect information games, such as poker and bridge, entail scenarios where players have only partial information about the game state [14]. Despite successes in games like poker [15], significant challenges persist, such as about hidden information based on the game's history [11].

Another important distinction in game complexity is between single-agent and multi-agent environments. Single-agent RL involves a single agent interacting with the environment to optimize its behavior, with notable success in games such as chess and go. Multi-Agent reinforcement learning (MARL) involves multiple agents, each influenced by the actions of others within a shared environment [16]. A notable example of MARL's potential was demonstrated in 2018 when OpenAI Five achieved superhuman performance in Dota 2 by defeating the world champions [17]. MARL algorithms generally fall into three categories: fully cooperative, fully competitive, and mixed. In cooperative settings, agents work together to maximize a shared long-term return. In competitive settings, agents play against each other and the returns typically sum to zero. Mixed settings involve both cooperative and competitive agents, with generalcsum returns [16]. In 2020, the Hanabi Challenge featured a state-of-the-art agent designed to play the purely cooperative game Hanabi [18].

One game that presents many challenges from an RL perspective is Tichu. Tichu is a four-player card game characterized by imperfect information, where teams of two players compete against each other. This results in a mixed setting, incorporating both cooperative and competitive elements. In 2020, Müller [11] applied RL to the task of playing Tichu, employing PPO along with imitation learning and other optimization techniques to develop an agent capable of playing the game at a decent human level. There has also been work on a simplified version of tichu called mini-tichu using monte-carlo tree search [19] and finite state machines coupled with search based methods [20] to create strong AI players.

The primary goal of this thesis is to develop a RL agent that is capable of playing Tichu at an amateur level, tackling the complexities of this strategic, imperfect information game. To achieve this, two distinct RL approaches will be employed and their performance evaluated against each other to determine which is better suited for a complex card game like Tichu. In particular, we will implement DQN [7], a value-based RL method that estimates the value of state-action pairs to guide decision-making, and PPO [8], a policy-based RL method that optimizes the policy directly through gradient ascent. Furthermore, additional optimizations, such as Behavioral Cloning (BC), will be incorporated to enhance the performance of the agents. The effectiveness of each approach will then be evaluated by comparing the agents against established baselines and against each other to assess their performance. Through this research, we hope to contribute valuable insights into the application of RL methods to multi-agent, imperfect information games, and to provide a foundation for future advancements in the field.

The following chapter provides an introduction to the game of Tichu, emphasizing the challenges it presents from a RL perspective and detailing the implementation of the Tichu environment. Chapter 3 outlines the algorithms and methods employed to develop the Tichu agents, along with the optimizations applied to enhance their performance. In Chapter 4 we describe the experimental setup used to evaluate our agents, present the results of our empirical evaluation, and offer a discussion of these results along with key insights gained. Finally, Chapter 5 concludes the thesis by summarizing our findings and providing an outlook on potential future work in this area.

Chapter 2

Tichu

This chapter provides an overview of the card game Tichu. Section 2.1 delves into the rules and the progression of the game. Section 2.2 explores the challenges that Tichu presents from a reinforcement learning (RL) perspective. Finally, Section 2.3 will discuss the Tichu environment and the implementation details of the game.

2.1 Tichu

Tichu is a card game created by Urs Hostettler in 1991 and published by Fata Morgana Spieleverlag. The game is typically played with four players, a version also known as Tichu Nanking. While there are other variants such as Threechu for three players and Tichu Tientsin for six players, these are rarely played [21]. This thesis focuses exclusively on the four-player version of Tichu Nanking, which will be referred to as Tichu from this point forward.

In the game of Tichu, players compete in teams of two, with partners positioned opposite each other and tasked with accumulating points collectively. The game employs a standard deck of 52 cards, comprising four suits each with 13 ranks, similar to the Western bridge pack. In this deck, the ace is the highest card and the 2 is the lowest. Moreover, there are four special cards — the Dragon, the Phoenix, the Hound, and the Mah Jong — that supplement the tichu deck [21].

At the start of the game, each player is dealt eight cards. Subsequently they may declare a 'Grand Tichu', whereby they wager that they will be the first to play all their cards. A successful outcome yields 200 points, whereas an unsuccessful one results in a loss of 200 points. Subsequently, the remaining cards are distributed, allocating 14 cards to each player. Prior to executing their initial play, participants may initiate a "Tichu," a parallel wager to the "Grand Tichu," offering a payout

of 100 points for a favorable outcome and a loss of -100 points in the event of an unfavorable result.

Once all participants have received their 14 cards, the trading phase commences. During this phase, each player engages in a series of secret exchanges, trading one card with each of the other players. Once the exchanges are complete, the player holding the Mah Jong card leads the first trick by playing a combination. The combinations in Tichu are similar to those in poker, with some variations. The possible combinations include single cards, pairs, trios, full houses, straights (five or more cards), pair steps (consecutive pairs), and bombs (four of a kind or straight flush).

During a trick, each player has to option to either pass or play a higher combination of exactly the same type as the current combination. The only exception to this rule is the playing of bombs, which can be played at any point in the game and do not need to match the current combination. A trick ends when three consecutive players pass. In this case, the player who played the last combination wins the trick. This player then leads the next trick with a new combination of their choice.

The round is concluded when only one player has cards remaining or when both players of a team have finished in the first and second position, respectively. In the event of a double victory, the team is awarded 200 points. In the event that a double victory does not occur, the accumulated points of the players are tallied. The player who finished last must relinquish their remaining cards to the opposing team, and the tricks they have won to the player who finished first. Subsequently each player tallies their points: king and tens are worth 10 points each, fives are worth five points each, the dragon is worth 25 points, and the phoenix is worth -25 points. All other cards are worth zero points. Furthermore, points from 'Grand Tichus' and 'Tichus' are added or subtracted, depending on their outcome, with ± 100 and ± 200 points respectively. The game continues until one team reaches or exceeds a score of 1000 points, thereby winning the game.

2.2 Challenges

Tichu is a complex game for humans due to its unique elements, including special cards, the trading phase, the announcement of 'Tichus' and 'Grand Tichus', and the scoring system. Furthermore, several challenges emerge when approaching Tichu from a RL perspective.

- **Imperfect Information:** Tichu is a game characterized by imperfect information, as only part of the game state is visible to all players. In particular, players are unaware of the cards that their opponents are holding. This lack of information makes it challenging for players to make informed decisions, as the optimal move can vary significantly depending on the unknown cards held by the other players.

- **Multi Agent:** Tichu is played in two teams of two, thereby creating a multi-agent RL setting. In this environment, the actions of each agent influence the actions of the other agents, which significantly increases the difficulty of identifying an optimal policy compared to single-agent settings.

- **Competitive and Cooperative:** Tichu features a mixed setting of cooperative and competitive play. Players must cooperate with their teammate, assisting them in discarding their cards and scoring points, particularly if they have announced a 'Tichu' or 'Grand Tichu'. Conversely, they must compete against their opponents, preventing them from discarding their cards and scoring points.

- **Sparse and Late Rewards:** Tichu rounds typically last between 70 and 120 turns, and the final score, or reward, is only revealed at the end of the round [11, 19]. This presents a challenge known as the credit assignment problem, wherein the agent must determine which actions taken throughout the round influenced the final reward. Furthermore, there exists a discrepancy between long-term and short-term rewards. For instance, winning a high-value trick does not guarantee that the player will retain those points, as they could be transferred to the winning team if the player finishes last.

- **Large State and Action Space:** Tichu features an extensive state and action space, encompassing more than 10^{30} possible hand distributions and allowing for approximately 10^{9} combinations to be formed from 14 cards [11].

- **Strategic Depth:** Tichu comprises several phases, including the announcement of Tichus, the trading of cards, and the playing of combinations. Each of theses phases presents a distinc set of challenges for RL agents. Additionally, as the game progresses, players must adapt their strategies in response to the evolving circumstances. For instance, if a team is trailing by a significant margin, they may need to take greater risks, potentially announcing a "Tichu" or even a "Grand Tichu.".

2.3 Tichu Environment

For our Tichu environment, we utilized an existing implementation by Müller, based on his work on developing a Tichu bot [11]. Müller's implementation itself was derived from a GitHub repository [22], for which no written documentation or results could be found online. In the following section, we will discuss some interesting aspects of the Tichu environment.

To enhance training performance, some simplifications were made to the original Tichu rules. We eliminated straight bombs, reducing the action space from approximately 10^9 to 10^3 [11]. This simplification was made possible by the fact that card color is only relevant when playing a straight bomb combination. Therefore by removing straight bombs, it is possible to ignore the color. Furthermore, asynchronous bombs were disabled to streamline the turn-taking process. With asynchronous bombs, it would be necessary to ask all players if they wanted to play a bomb after every move, significantly increasing the episode length. Finally, we defined a single round of Tichu as one episode, rather than playing until 1000 points, to further shorten the episode duration.

To train the RL agent on Tichu, the environment must encode the current game state for the agent. We accomplish this by creating a vector of input features derived from the observable state of the game. Examples of these features include the player's hand cards, the number of cards remaining for all players, the current trick on the table, and whether 'Tichus' and 'Grand Tichus' have been announced. In addition to these observable features, we also include some historical game data. Since a typical round lasts about 70 to 120 moves, it is impractical to include every move as this would result in an excessively large input vector. Instead, we use handcrafted features that summarize the history, such as cards already played and information about what combinations other players have passed. This approach results in a state vector with a total of 85 features.

The action space in our environment includes all legal actions possible in the game of Tichu. This includes not only all valid combinations that can be made with 14 cards but also other actions such as passing and calling Tichu. Thus, there are 1,415 different actions in Tichu. In order to distinguish between legal and illegal actions in a given state of the game, we provide a legal-actions mask that indicates which actions from the action space are allowed in the current state.

We defined the reward as the score at the end of the round. Specifically, we calculated the difference between the points scored by the agent's team and the points scored by the opponent's team. This approach aims to maximize the point differential, not than just the points scored by the agent's team. Consequently, our agent is incentivized to prevent the opposing team from successfully completing a Tichu, which would otherwise only reduce the opponent's score without directly benefiting the agent's team's score.

Chapter 3

Reinforcement Learning

This chapter introduces the reinforcement learning (RL) algorithms and techniques used in this thesis. We will first outline the basic concepts of RL, and then proceeds to examine two distinct approaches that will be empirically evaluated. Each subsections will include a paragraph detailing the implications of the specific algorithms and techniques for the Tichu card game.

3.1 Reinforcement Learning

RL is an important subfield of machine learning, alongside supervised learning and unsupervised learning. In supervised learning, the objective is to learn a function that maps inputs to outputs based on sample input-output pairs. This function is then inferred from labeled training data, which consists of a set of input-output pairs [23]. In contrast, in unsupervised learning, there are no correct outputs or labels within the data set. Instead, the algorithms are tasked with independently discovering patterns and structures within the data, making this approach primarily useful for clustering and feature reduction. In contrast, RL is concerned with training an agent to make a series of decisions within an environment with the goal of maximizing a cumulative reward.

The basic elements of RL are the agent, the environment, actions, states, and rewards. The agent interacts with the environment and learns from these interactions. By taking actions within the environment, the agent receives rewards and new states as feedback. The environment is the context in which the agent operates and provides rewards based on the agent's actions. The state reflects the agent's current situation within the environment, including relevant information that may influence the agent's decision-making process. Actions are the decisions made by the agent that affect the state of the environment and can lead to rewards. Rewards

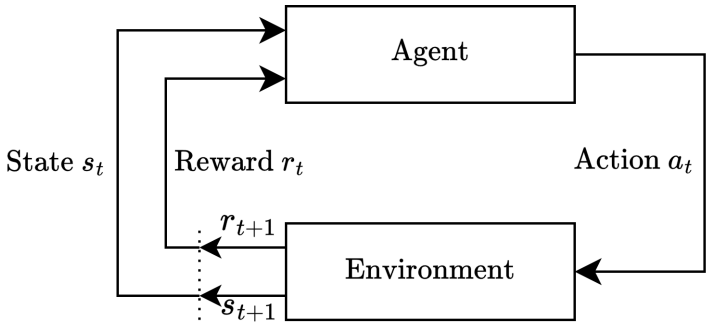


Figure 3.1: Basic RL ,odel illustrating agent-environment interactions.

are the positive or negative feedback the agent receives from the environment based on its actions. The goal of RL is to maximize cumulative rewards over time, aiming to discover an optimal policy that yields the highest possible long-term rewards.

The training of an RL agent is an iterative process as illustrated in Figure 3.1. The agent commences in an initial state s_0 and selects an action a_0, which is then executed by the environment. This action causes the environment to transition to a new state s_1 and potentially provides a reward r_1 corresponding to the chosen action. The agent then selects the next action a_1, and the cycle continues. Through this process, the agent continuously interacts with the environment, receiving feedback for its actions. This iterative feedback loop allows the agent to refine its policy over time, aiming to achieve higher cumulative rewards.

Formally, according to Huys [5], RL addresses solutions to Markov Decision Process problems, which are characterized by the tuple $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}$, and π:

- $\mathcal{S}$: set of states $s \in \mathcal{S}$
- $\mathcal{A}$: set of actions $a \in \mathcal{A}$
- $\mathcal{T}(s'|s,a)$: the transition function mapping each state-action pair to a distribution over successor states s', with $s, s' \in \mathcal{S}, a \in \mathcal{A}$, and $\sum_{s'} \mathcal{T}(s'|a,a) = 1$
- $\mathcal{R}(s,a,s') \rightarrow r$ **:** the reward function mapping state-action-successor state triples to a scalar reward r
- $\pi(s) \rightarrow a$ **:** the policy maps each state to an action that should be taken

The objective is to find an optimal policy $\pi(s) \rightarrow a$ that maps each state to the optimal action a^*, maximizing the total future return of actions a in state s:

$$a^* \leftarrow \arg\max_a \mathcal{Q}(s,a) \quad \text{where} \quad \mathcal{Q}(s,a) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s, a\right]$$

The Q-function, $\mathcal{Q}(s,a)$, represents the expected total return of taking action a in

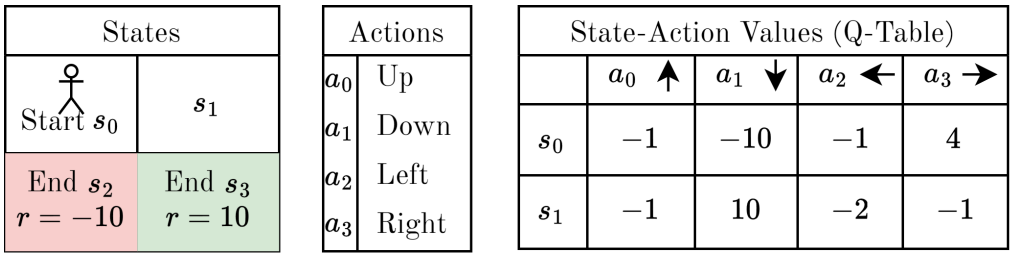


Figure 3.2: Simple example of a RL problem

state s and following the optimal policy thereafter. The discount factor $0 \leq \gamma \leq 1$ determines the importance of immediate versus future rewards. When $\gamma = 0$ only the next reward is considered, whereas when $\gamma = 1$ all future rewards are treated with equal importance, regardless of when they occur.

Simple RL example Figure 3.2 provides a simple RL example. The agent commences in state s_0 and can move according to actions a_0 to a_3. Should the agent arrive at s_2 or s_3, a reward of -10 or 10 respectively, is conferred; otherwise, the reward is zero. The table on the right shows an example of the values for state-action pairs. From these values, we can infer that the optimal policy for our agent is to take action a_3 (right) in state s_0 and then action a_1 (down) in state s_1. This policy results in the highest possible reward of r=10.

As previously stated in the introduction, RL can be broadly categorized into two main approaches: model-based and model-free.

Model-based RL postulate that the transition matrix $\mathcal{T}$, reward function $\mathcal{R}$, and the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ are known, thereby defining the model of the environment. This implies that the agent has comprehensive knowledge of its operating environment. The objective of the agent is to ascertain an optimal policy for transitioning from the current state to the goal state.

In contrast, model-free RL methods are well-suited for scenarios where the agent lacks knowledge about the environment, particularly the transition matrix $\mathcal{T}$ and the reward function $\mathcal{R}$. The agent acts based on its current state without knowing the future outcomes of its actions. Instead, it relies on the rewards received from taking actions to learn about the environment. Consequently, model-free methods employ a trial-and-error approach to discover and optimize the policy.

Implications for Tichu: Tichu is not a suitable candidate for either supervised or unsupervised learning due to its dynamic, sequential nature and the necessity for goal-oriented, reward-based learning, which is better handled by RL. Tichu is a game of imperfect information with unpredictable actions from other players, which renders model-based RL impractical. Instead, we employ model-free RL to find an optimal policy through trial and error. Section 2.3 detailed the implementation of the environment, states, actions, and rewards. The subsequent sections will examine the techniques employed for our agents, with a particular focus on the value-based method Deep Q-Networks (DQN) and the policy-based method Proximal Policy Optimization (PPO).

3.2 Value Based Reinforcement Learning

Value-based methods estimate the value (expected return) of being in a given state. If all state-action values are known, the optimal action can be selected, as illustrated in Figure 3.2. In model-based RL, these values can be explicitly calculated using the Bellman equation: [24]

$$\mathcal{Q}(s,a) = \sum_{s'} \mathcal{T}(s'|a,s)\left[\mathcal{R}(s,a,s') + V(s')\right] \quad \text{with} \quad V(s') = \max_{a'} \mathcal{Q}(s',a') \tag{3.1}$$

The Q-value of a state-action pair represents the long-run expected return for taking the action a in the state s. The optimal policy is defined as the mapping of each state to the action with the highest Q-value: $\pi^* \leftarrow \arg\max_a \mathcal{Q}(s,a)$.

In model-free RL, this is not a viable approach since the transition matrix $\mathcal{T}$ and the reward function $\mathcal{R}$ are unknown. This impedes the precise calculation of Q-values for all state-action pairs. Consequently, we approximate the Q-values through sampling from the environment.

We can further distinguish between off-policy and on-policy RL methods. In on-policy RL, actions are generated in response to observed states from the environment according to a specific policy. The outcomes of these actions are then used to iteratively refine the parameters of the same policy. In on-policy RL, the same policy is used both for exploring the environment (behavior policy) and for optimizing the learning objective (target policy). This dual role often requires the incorporation of some level of randomness in the action selection to effectively balance exploration and exploitation.

In contrast, off-policy RL decouples the data collection process from the policy training process by maintaining two distinct policies: a behavior policy and a target

policy. The behavior policy generates actions within the environment, while the target policy is trained iteratively using the resulting data. This decoupling enables off-policy RL to leverage previously collected data (or data from other policies) to enhance the target policy, thereby facilitating greater flexibility in learning from disparate sources of experience.

Implications for Tichu As previously stated, we opted for a model-free RL approach for our Tichu agents. In regard to the policies that direct the actions of these agents, both on-policy and off-policy algorithms are viable options. We utilized DQN which is an off-policy, value-based method. In contrast, our policy-based method PPO will adopt an on-policy approach. This decision is consistent with the typical use of on-policy algorithms in policy-based methods, thereby providing an opportunity to compare the performance of on-policy versus off-policy strategies across our agents.

3.2.1 Q-learning

Q-learning is a type of value-based RL. During the learning process, the agent selects an action in a given state and evaluates its consequences based on the immediate reward or penalty received, as well as the estimated value of the resulting state. By repeatedly exploring all possible actions across different states, the agent gradually learns which actions yield the highest long-term discounted rewards [6].

To update the Q-value for a given state-action pair, we use the following equation:

$$\underbrace{\mathcal{Q}^{\text{new}}(s,a)}_{\text{new Q-value}} = \underbrace{(1-\alpha)\cdot\mathcal{Q}(s,a)}_{\text{old Q-value}} + \underbrace{\alpha\Big(\mathcal{R}(s,a,s') + \gamma \max_{a} \mathcal{Q}(s',a)\Big)}_{\text{learned Q-value}} \tag{3.2}$$

This equation modifies the Q-value $\mathcal{Q}(s,a)$ by blending the current Q-value with new information gained from recent experiences. The learning rate α controls the weight given to the newly learned value versus the old estimate, while the discount factor γ adjusts the significance of future rewards in the update process.

In the case of non-deep learning approaches, the Q-function is represented in tabular form, which is commonly referred to as a Q-table. An illustrative example of this representation can be found in Figure 3.2. The attainment of a comprehensive Q-table encompassing all state-action pairs would signify optimal gameplay and the resolution of the game, as it would elucidate the most advantageous action for each state, that is, the one with the highest value. In practice, this fundamental approach

is inapplicable due to the inability to generalize across states and actions, as the action-value function is estimated independently for each sequence. Consequently, function approximators are commonly employed to estimate the action-value function, $\mathcal{Q}(s, a; \theta) \approx \mathcal{Q}^*(s, a)$. This enables effective generalization and the handling of large state-action spaces [25].

Implications for Tichu The Q-learning method can be implemented by sampling actions from the environment and computing Q-values. However, given the large state and action space in Tichu, constructing a Q-table for all state-action pairs is impractical. Therefore, a Q-function is used to approximate these values. Neural networks, which are known for their effective function approximation capabilities, are employed in this context. Consequently, Deep Q-learning is used, which will be detailed in the following subsection.

3.2.2 Deep Q-Networks

Deep Q-learning represents an extension of Q-learning, employing neural networks as function approximators. A Deep Q-Network (DQN) is a multi-layered neural network that, given a state s, outputs a vector of action values $\mathcal{Q}(s, \cdot; \theta)$, where θ represents the network parameters [7]. This neural network, referred to as the Q-Network, maps states to action values and is utilized for training the agent. In contrast, a second neural network, designated as the Target Network, with parameters θ', is employed to evaluate the Q-Network's performance and stabilize training. The use of these two networks is known as Double Q-Learning [7].

In addition to the two neural networks, an experience replay buffer is employed, which interacts with an environment simulator to gather training data. The experience replay buffer employs the ε-greedy strategy for the selection of actions. In this approach, a threshold value $0 \leq \varepsilon \leq 1$ is predefined. In the event that a random number between 0 and 1, denoted by p, exceeds the threshold value $p \geq \varepsilon$, the agent selects the action with the highest current value, $a = \arg\max_a \mathcal{Q}(s, a)$. Otherwise, a random action is chosen. This approach represents a compromise between exploitation, which makes use of known strategies, and exploration, which investigates potentially superior, hitherto unknown actions. Subsequently, the agent executes the selected action a, receives a reward r, and transitions to the subsequent state s' from the environment simulator. This tuple of state, action reward and next state is then stored as a sample (s, a, r, s'), in the experience replay buffer for training purposes.

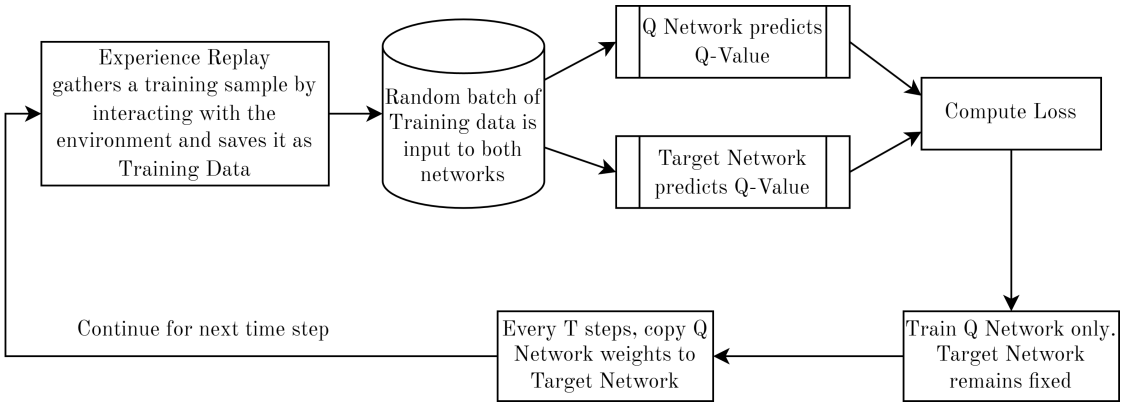


Figure 3.3: High-level DQN workflow

Figure 3.3 offers a comprehensive overview of the DQN workflow. In the initial stage, the experience replay buffer gathers training samples as previously described. Subsequently, a random batch of these samples, comprising a mix of older and more recent data, is then selected for training. The aforementioned batch is then provided as input into both the Q-Network and the Target Network. The Q-Network processes each sample's current state s and action a in order to predict the Q-value of that action. Subsequently, the predicted Q-value is then compared with the target Q-value, which is derived by the Target Network using the next state s' and estimating the highest Q-value across all possible actions from that state. The loss is then calculated using the predicted Q-value, the target Q-value, and the observed reward, which is then employed to update the Q-Network through backpropagation. Throughout this process, the Target Network remains unaltered.

The process then continues with the subsequent time step. During this phase, only the weights of the Q-Network are updated, while those of the Target Network remain unaltered. This configuration permits the Q-Network to refine its predictions of Q-values, while the Target Q-values remain constant for a designated period. After a predetermined number of time steps, the Q-Network's weights are copied to the Target Network. This update ensures that the Target Network also benefits from the improved weights, enabling it to predict more accurate Q-values. Training then resumes as before.

Implications for Tichu In order to implement DQN for Tichu, the DQN implementation from Stable-Baselines [26] is employed. The training data for the experience replay is generated using the Tichu environment described in Section 2.3.

3.3 Policy Based Reinforcement Learning

Policy-based methods do not rely on maintaining a value function model; rather they focus on directly identifying an optimal policy π^*. Typically, a parameterized policy π_θ is used, wherein its parameters are adjusted to maximize the expected return $\mathbb{E}[R|\theta]$ through either gradient-based or gradient-free optimization techniques [2]. When constructing a policy directly, it is common to generate parameters for a probability distribution, resulting in a stochastic policy from which actions can be sampled directly.

Implications for Tichu When applying policy-based RL to Tichu, it's important to consider that Tichu is a stochastic environment. Consequently, we cannot directly compute the trajectory for each policy π_θ to find the one that yields the highest cumulative reward. Instead, trajectory probabilities are influenced by the policy but are not entirely determined by it.

3.3.1 Policy Gradient Method

Policy gradients are preferred for their effectiveness in improving a parameterized policy, as they provide a stable learning signal. In order to calculate the expected return $\mathbb{E}[R|\theta]$ of a policy π_θ, it is necessary to average over the various trajectories induced by the current policy parametrization. These trajectories, denotes as $\tau = (s_1, a_1, ...s_t, a_t)$, consist of sequences of states and actions within an episode, determined by the policy. Each trajectory has an associated probability $P(\tau)$ and a cumulative reward given by $R(\tau) = \sum_t \gamma^t R_t$ where the rewards R_t are discounted by γ.

Therefore, our goal can be formalized as maximizing the expected reward over time. In this context, $\tau \approx \pi_\theta$ represents the distribution of trajectories generated by the current policy. We can alternatively express this objective by summing over all trajectory probabilities and weighting them by their corresponding rewards. The objective function $J(\theta)$ is defined as follows: [27]

$$J(\theta) = \mathbb{E}_{\tau \approx \pi_\theta} R(\tau) = \sum_\tau P(\tau; \theta) \cdot R(\tau) \tag{3.3}$$

The corresponding maximization problem is represented by:

$$\max_\theta J(\theta) = \max_\theta \sum_\tau P(\tau; \theta) \cdot R(\tau) \tag{3.4}$$

The maximization problem demonstrates that modifying the value of θ affects the probabilities of trajectories. As previously discussed, in model-free RL, due to the stochastic nature of the environment, trajectories are influenced by but not solely dictated by the policy. Consequently, two probability distributions are considered. The policy $\pi_\theta(a|s) = P(a|s;\theta)$ represents the probability of selecting each action in a given state and is dependent on the parameter θ. In contrast, the transition probability distribution $P(s_{t+1}|s_t, a_t)$ specifies the likelihood of reaching state s_{t+1} after performing action a_t in state s_t.

By emplyoing these two probabilities, it is possible to ascertain the probability of a trajectory τ occurring under the policy π_θ. For each time step, we multiply the probability of selecting an action by the transition probability. By combining these probabilities across all time steps, we obtain the overall probability of the entire trajectory:

$$P(\tau;\theta) = \left[\prod_t \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{transition function}} \cdot \underbrace{\pi_\theta(a_t|s_t)}_{\text{Policy}}\right] \tag{3.5}$$

To maximize our expected reward $J(\theta)$, we then compute the gradient of the expected reward with respect to θ:

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \tag{3.6}$$

Using the properties of logarithms we can rewrite the expression to:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau) = \mathbb{E}_{\tau \approx \pi_\theta} \nabla_\theta \log P(\tau;\theta) R(\tau) \tag{3.7}$$

Therefore we can also write the gradient as an expactation which is crucial in applying sampling-based methods.

By substituting the trajectory probability from equation 3.5 and applying logarithms, it is possible to separate the transition function from the policy. This allows us to exclude the transition function from the gradient calculation, as it is independent of θ:

$$\nabla_\theta \log P(\tau;\theta) = \nabla_\theta \left[\sum_t \log P(s_{t+1}|s_t, a_t) + \sum_t \log \pi_\theta(a_t|s_t)\right] = \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{3.8}$$

Equation 3.8 represents the true expectation over all possible trajectories. Given that typical RL problems are computationally intractable, we work with a finite sample of trajectories instead. Consequently, we approximate the gradient based on these samples. Since the gradient itself is an expectation, it can be estimated through simulation:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P\left(\tau^{(i)}; \theta\right) \cdot R\left(\tau^{(i)}\right) \tag{3.9}$$

This expression is fully tractable because we can sample the trajectories and their corresponding rewards can be sampled without the need to know the environment model. The only requisite is a policy that is explicitly defined and differentiable with respect to θ.

Ultimately, in order to update the policy parameters we define a learning rate $0 < \alpha \leq 1$, indication the weight placed on the computed gradient in order to update the existing parameters. The corresponding update rule is as follows:

$$\theta \leftarrow\!\shortmid \theta + \alpha \nabla_\theta J(\theta) \tag{3.10}$$

Implications for Tichu Selecting an appropriate learning rate α represents a significant challenge, as it can result in either over- or undershooting, and consequently, missing the reward peak or stalling prematurely. Furthermore, due to the potential for substantial policy changes, samples are typically used only once. This necessitates re-sampling with the updated policy, which can be inefficient, particularly in complex environments like Tichu. Additionally, reward trajectories can vary considerably, leading to unstable updates. Consequently, we will utilise PPO, an enhanced policy-based method.

3.3.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO), introduced by John Schulman in 2017 [8], is a policy gradient method designed to enhance the stability of policy training. The fundamental concept of PPO is to limit the scope of policy changes during each training iteration. By constraining these updates, PPO facilitates the prevention of excessive modifications to the policy, which can result in instability and suboptimal performance. Smaller, controlled updates enhance the probability of converging to an optimal solution, whereas larger updates increase the risk of overshooting and destabilizing the training process.

In our objective function, we frequently employ an advantage function A rather than the reward function R discussed previously. The advantage function is defined as $A(s,a) = Q(s,a) - V(s)$, where $Q(s,a)$ represents the value of taking action a in state s, and $V(s)$ is the value of the state s. The advantage function quantifies how much better an action a is compared to the average value of being in state s. If $A(s,a) > 0$, it indicates that action a is more favorable than other possible actions in state s. Consequently, the policy objective function can be framed using this advantage function as follows:

$$L^{\mathrm{PG}}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t) \cdot \hat{A}_t)] \tag{3.11}$$

Here, $\hat{A}_t$ is an estimator of the advantage function at timestep t. When multiple optimization steps are perfomed on the loss L^{PG} using the same trajectory, it often results in destructively large policy updates [8].

To avoid large policy updates we first define the probaility ratio $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ which is defined as the probability of taking action a_t in state s_t under the current policy divided by its probability under the old policy. If $r_t(\theta) > 1$, it indicates that the action a_t at state s_t is more likely under the current policy compared to the old one. Conversely, if $r_t(\theta) < 1$, the action is more likely under the old policy. This probability ratio helps estimate the divergence between the old and current policies.

Using this ratio PPO defines the objective function as follows: [8]

$$L^{\mathrm{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min\Big(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\Big)\right] \tag{3.12}$$

where ε is a hyperparameter that controls the extent to which the policy can change. The first term inside the min function is the surrogate objective $r_t(\theta)\hat{A}_t$, which represents the likelihood of an action, scaled by the advantage of the action relative to other actions. The second term clips this probability ratio to the range $[1-\varepsilon, 1+\varepsilon]$. Ultimately, taking the minimum of the clipped and unclipped objectives ensures that the final objective is a lower bound on the unclipped objective. Consequently, changes in the probability ratio are ignored if they might improve the objective, but they are considered if they might worsen it.

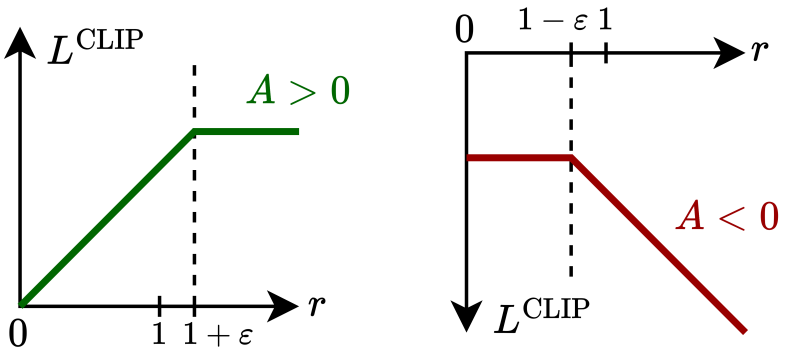


Figure 3.4: Visualization of the Clipping in PPO

Figure 3.4 illustrates the impact of clipping in PPO. In the event that $r_t(\theta)$ falls within the range $[1-\varepsilon, 1+\varepsilon]$, the objective remains unclipped and is simply $r_t(\theta)\hat{A}_t$, irrespective of the advantage estimate. However, if $r_t(\theta) < 1-\varepsilon$, indicating that the action is much less likely under the current policy than under to the old one, the objective will be clipped if the advantage estimate is negative. This prevents the probability of an unfavorable action that is already less likely from being further reduced. Conversely, if $r_t(\theta) > 1+\varepsilon$, indicating that the action is more likely under the current policy, the objective will be clipped if the advantage estimate is positive. This helps to avoid excessive increases in the probability of actions that are already favored by the current policy.

Implications for Tichu In order to implement PPO for Tichu, the PPO2 algorithm from the Stable-Baselines library [28] is employed. In contrast to PPO, PPO2 has been designed for distributed training, with one process responsible for training the policy network and multiple processes used to simultaneously run the environment and collect training data in parallel.

3.4 Imitation Learning

Imitation learning (IL) is a field of study that aims to replicate human behavior for specific tasks. The agent learns to perform actions based on demonstrations by mapping observations to actions. This approach involves analyzing the behavior of experts and the environment. As with traditional supervised learning, where examples consist of feature-label pairs, IL deals with pairs of states and actions. IL can generally be divided into two main approaches: behavioral cloning and inverse reinforcement learning [29].

Implications for Tichu The applocation of IL to Tichu is advantageous due to the game's inherent complexity and strategic depth. IL facilitates accelerated learning and effectively addresses the game's intricate decision-making processes with reduced data.

3.4.1 Behavioral Cloning

In the field of Behavioral Cloning (BC), we utilize supervised learning to develop a policy based on expert demonstrations. Initially, we collect these expert trajectories from human players or other agents. Subsequently, we pretrain our network using these demonstrations and then refine the policy network to accurately replicate the expert behavior accurately.

Implications for Tichu To apply BC we employed the pretrain method from the Stable-Baselines library [30]. Our expert trajectories were obtained by scraping game logs from the online board-game platform Brettspielwelt [31]. We focused exclusively on the state-action pairs from the winning team for each round. Games featuring straight or asynchronous bombs were excluded as they are not compatible with our environment.

Chapter 4

Empirical Evaluation

This section presents an empirical evaluation of the agents, comparing them against baseline models and each other. It begins by detailing the agents and their training processes. It then discusses the challenges encountered during the evaluation of the agents, how we addressed these challenges, and the experimental setup. Finally, it presents the evaluation results and analyzes notable insights about the agents.

4.1 Evaluation

In our evaluation, we employed four distinct agents: Deep Q-Network (DQN), DQN with Behavioral Cloning (BC), Proximal Policy Optimization (PPO), and PPO with BC. For pretraining, we used an expert dataset comprising four million actions. The optimal dataset size was determined to be four million actions, as larger datasets, such as those with ten or twenty million actions, resulted in poorer performance, likely due to overfitting. The policy network was pretrained for ten epochs using the entire expert dataset before transitioning to standard training. Subsequently, each agent was trained for 3,000 epochs, with each epoch consisting of 50,000 steps, totaling 150 million steps.

The agent is trained using a shared policy network for both players on the agent's team. We implemented opponent sampling [32], where the agents compete against the most recent policy for 80% of their games and against randomly selected older policies for the remaining 20%. This approach ensures that the agents do not become overly specialized in countering only the current policies but can also perform well against a variety of strategies.

The training of our agents exclusively conducted using CPUs due to the high time complexity of the Tichu environment. The training was carried out on UBE-

LIX, the high-performance computing cluster at the University of Bern. By leveraging 32 CPUs, we were able to achieve a training speed of approximately 750 million steps per hour, which is equivalent to around 18 million steps per day.

4.1.1 Experimental Setup

The evaluation of a Tichu agent's performance presents a significant challenge, primarily due to the inherent luck factor in the game. For example, the initial distribution of hand cards can have a substantial impact on the outcome of a round, with even the most skilled players potentially losing to those with less experience if they are dealt unfavorable hands. To ensure a fair and accurate evaluation, it is essential to mitigate the effects of this randomness.

In order to mitigate the impact of card luck, a fixed set of hand cards is generated for all evaluations. Each set of hand cards is played twice: first with the cards dealt normally, and then with the cards shifted one position to the right. This shifting cancels out any advantages that may be gained from particularly good hands.

Furthermore, the agent's reward was averaged over 10,000 rounds of Tichu. As described in Section 2.3s, the reward is calculated as the point difference between the teams. Consequently, when competing against an opponent of an equivalent strength, the expected reward would be zero.

4.1.2 Baselines

In order to evaluate the performance of our agents, two additional baseline models were employed: a random agent and a rule-based agent. These handcrafted baselines serve as a fundamental reference point for gauging the effectiveness of our agents.

- **Random Agent:** In accordance with the rules of the game, the random agent selects an action at random from the set of legal actions on each turn. As this agent does not adhere to a defined strategy, it is anticipated that it will be outperformed by the other agents.
- **Rule Agent:** The rule-based agent is programmed to make decisions in accordance with the established Tichu strategies. At the start of a round, it refrains from calling 'Grand Tichu' or 'Tichu'. When trading cards, the agent consistently gives its two lowest cards to opponents and its highest cards to

its partner. During gameplay, the agent consistently plays the lowest possible combination. When leading a new trick, it opts for the combination with the highest number of cards. As a result, the agent can be considered comparable in strength to a novice who has just learned the game or an amateur familiar with basic Tichu tactics.

4.2 Results and Discussion

Table 4.1 presents the results of our evaluation. The rows represent the agents, while the columns indicate the opponents each agent faces. In each column, the best score is highlighted in bold for clarity.

	DQN	DQN/BC	PPO	PPO/BC	Random	Rule	Total	Rank
DQN[7]	-	-113	-154	-325	32	-197	-757	16
DQN[7]/BC[29]	113	-	-29	**-191**	168	-136	-75	11
PPO[8]	154	29	-	-223	257	-210	7	12
PPO[8]/BC[29]	**325**	**191**	**223**	-	**414**	**90**	**1243**	**5**

Table 4.1: Evaluation results
The "Total" column sums all the rewards accumulated by each agent, while the "Rank" column aggregates the rankings each agent achieved across all matchups.

4.2.1 Comparison of Models

The four agents tested were quite different in their performance:

- **DQN:** The DQN agent without BC performs poorly compared to all other agents, suffering significant losses, particularly against the PPO with BC agent. While it does show some improvement by defeating the Random agent, it still struggles against the rule-based agent. In summary, it attains the lowest total reward and rank index.

- **DQN with BC:** The pretraining of the DQN agent with BC resulted in a notable enhancement in its performance. The agent demonstrated a notable superiority over the standard DQN, achieving a substantial margin of victory. While the agent was outmatched by the PPO agent, it attained the highest reward against the PPO with BC agent.Additionally, it secured a higher reward against the rule agent than the PPO Agent, suggesting that the DQN

with BC agent may be more effective against more formidable Tichu players. Finally, it has the second worst total, but is the second best on the rank index.

- **PPO:** The PPO agent appears to demonstrate superior performance in training without BC compared to the DQN agent. The PPO agent outperforms the DQN and DQN with BC agents as well as the random agent. Notably, the PPO agent demonstrates the poorest performance against the rule agent. This indicates that the PPO agent may have identified a strategy that is effective against less skilled opponents, but is less optimal against more skilled ones. Despite having the second-highest total reward, the PPO agent is ranked third behind the DQN with BC agent in the ranking index.
- **PPO with BC:** The PPO agent with BC demonstrates superior performance compared to all other agents, exhibiting a significant advantage in both total reward and rank index. It successfully defeats every other agent, including the rule-based agent, which no other agent managed to do. In both total reward and rank index, the PPO agent with BC stands out as the clear leader, making it the most effective agent overall.

.

4.2.2 Evaluation of Actions

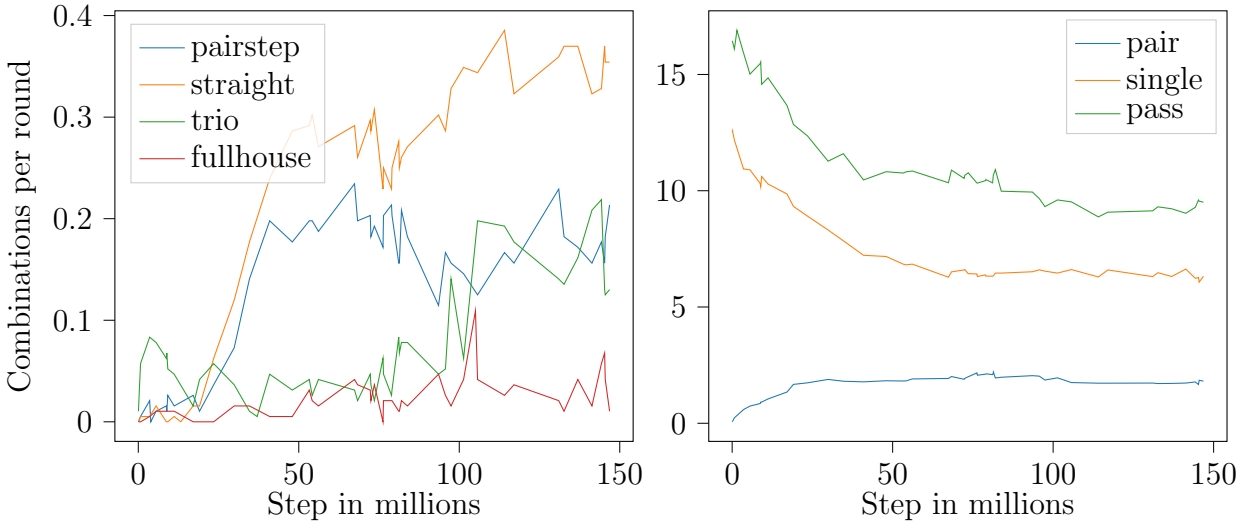


Figure 4.1: Combination frequeny PPO with BC

Figure 4.1 provides valuable insights into the training process of the PPO with BC agent, which has demonstrated the greatest performance. The combination frequencies for the other agents are provided in Appendix A. On the left, we observe

the frequency of longer combinations played per round. It is notable that the use of straights increased significantly during training, while the frequency of playing pair steps and trios also rose, though to a lesser extent. Full houses saw a modest increase. Overall, it is evident that the agent increasingly favored longer combinations as training progressed.

In contrast, the frequency of shorter combinations, such as singles, decreased, while pairs exhibited only a slight increase. Furthermore, the agent passed less frequently after training. This shift aligns with our expectation of an improving Tichu player, as playing longer combinations is generally more advantageous when possible.

Chapter 5

Conclusions and Future Work

This chapter concludes the thesis by revisiting the central research question. We begin by providing a summary of the methods employed and the results achieved. Next, we present and discuss the key findings, thereby providing an answer to our research question. Finally, in Section 5.2, we explore opportunities for future work and propose directions to further build upon the outcomes of this study.

5.1 Conclusions

The primary goal of this thesis is to develop a reinforcement learning (RL) agent that is capable of playing the card game Tichu at an amateur level. To achieve this, we explore two distinct RL methodologies: value-based Deep Q-Networks (DQN) and policy-based Proximal Policy Optimization (PPO). Furthermore, we integrate Behavioral Cloning (BC) to enhance our agents' performance using expert data. Following BC pretraining and agent training for 150 million steps, we assessed their performance through competitive matches against each other and two baseline agents. The four trained agents exhibited disparate performance levels.

The DQN agent demonstrated the least optimal performance across all evaluation metrics. This outcome may be attributed to the fact that the agent was primarily trained by playing against itself and earlier versions of itself. Consequently, it likely developed a policy that was effective against its own strategies but failed to perform well against other, stronger opponents.

However, the DQN with BC agent showed a marked improvement, outperforming the standard DQN agent in every metric. The BC appears to have helped the DQN agent develop a policy more effective against stronger players. Notably, the DQN with BC agent achieved the highest score against our strongest agent, PPO

with BC, and the second-highest score against the rule-based agent, only behind PPO with BC. In conclusion, at least in our configuration, DQN does not appear to be optimal choice for training agents to play Tichu effectively.

The PPO agent demonstrated superior performance compared to both the standard DQN and DQN with BC, suggesting that PPO is inherently more effective than DQN, even without the advantage of BC. However, despite this relative strength, the PPO agent still struggled against stronger opponents. It was defeated by the PPO with BC agent and exhibited the poorest performance among all agents when pitted against the rule-based agent.

In contrast, the PPO with BC agent demonstrated the greatest efficacy among all agents, achieving the best results across every evaluation metric. Moreover, it was the sole agent to prevail over the rule-based agent. This performance indicates that the PPO with BC agent has reached a level of proficiency comparable to that of an amateur player. This result is analogous to that observed with the DQN agent, where BC markedly enhanced the agent's ability to develop a more robust and adaptable strategy, particularly against stronger opponents, by leveraging expert data.

In conclusion, the results indicate that the combination of PPO with BC represents the most effective approach for training an RL agent to play Tichu. This finding suggests that the policy-based PPO method is inherently more suitable for this game than the value-based DQN method. Additionally, the incorporation of BC for pretraining the agents is recommended, as it significantly enhances their overall performance and adaptability, particularly against stronger opponents.

5.2 Future Work

The findings of this thesis indicate that RL agents are capable of learning and playing complex games like Tichu. However, further advancements are necessary for the agent to surpass human-level performance.

Building on this research, several potential avenues for future exploration in the realm of mastering Tichu, or other complex games, through RL have been identified. These include:

- **Evaluation against Humans** In this thesis, our agents were evaluated against each other and two baseline models. A natural progression would

be to assess their performance against human players. Evaluating the agents in matches with human opponents would provide valuable insights into their effectiveness and adaptability, offering a more comprehensive understanding of their capabilities in real-world scenarios.

- **Explainability and Interprebatilty** Although we were able to identify certain strategies employed by our agents through an analysis of the combinations they played, a comprehensive understanding of their decision-making processes remains elusive. Gaining insights into how the agents evaluate specific actions and the reasoning behind their choices would be invaluable. This could facilitate a more profound comprehension of their strategic approach and potentially illuminate the underlying mechanisms that drive their behavior.

- **Exploration of Tichu Variants:** While this thesis focused on the standard four-player Tichu variant known as Tichu Nanking, future research could explore other variants of the game. For instance, it would be of interest to investigate the performance of RL agents in the purely competitive three-player variant known as Threechu, or in Tichu Tientsin, a six-player variant where two teams of three compete against each other. These variations could present unique challenges and opportunities for refining RL strategies.

- **Transfer Learning** In addition to investigating Tichu variants, another promising avenue for future research is the application of transfer learning. By training RL agents on different card games or even entirely different types of games, researchers could investigate whether the methods effective for Tichu, such as PPO, are also applicable to other domains. This approach could reveal whether PPO remains the optimal choice across different games or if other RL methods might outperform it in new contexts.

Appendix A

Combination frequencies of agents

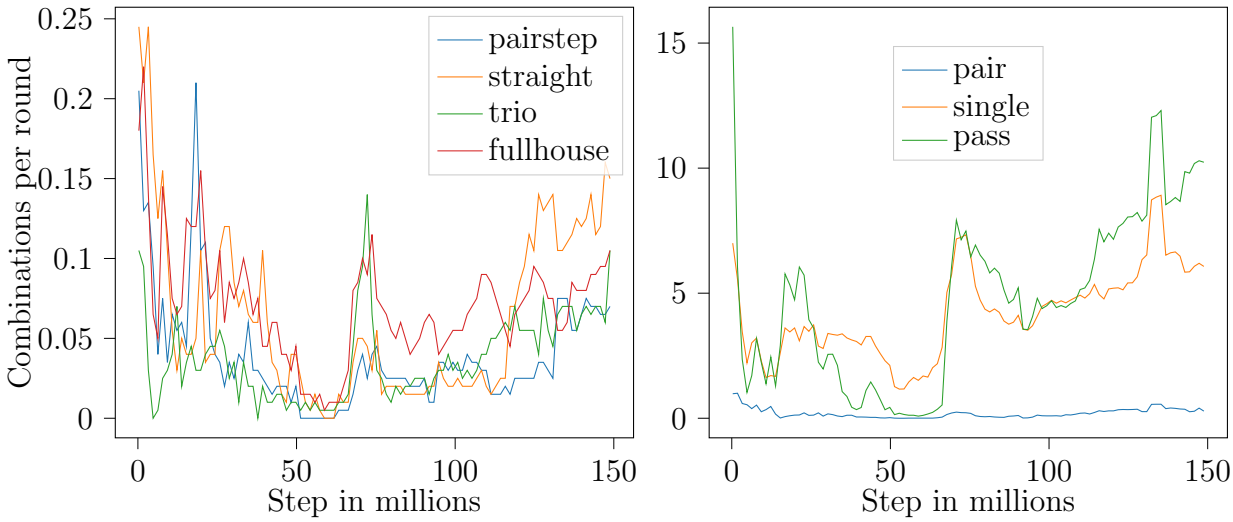


Figure A.1: Combination Frequeny DQN

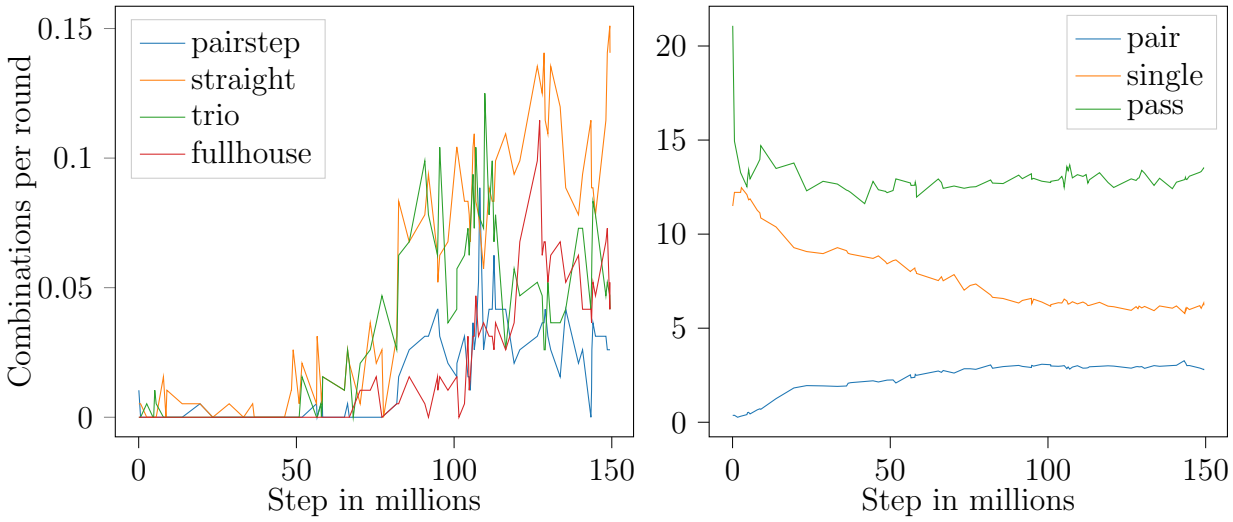


Figure A.2: Combination Frequeny DQN with BC

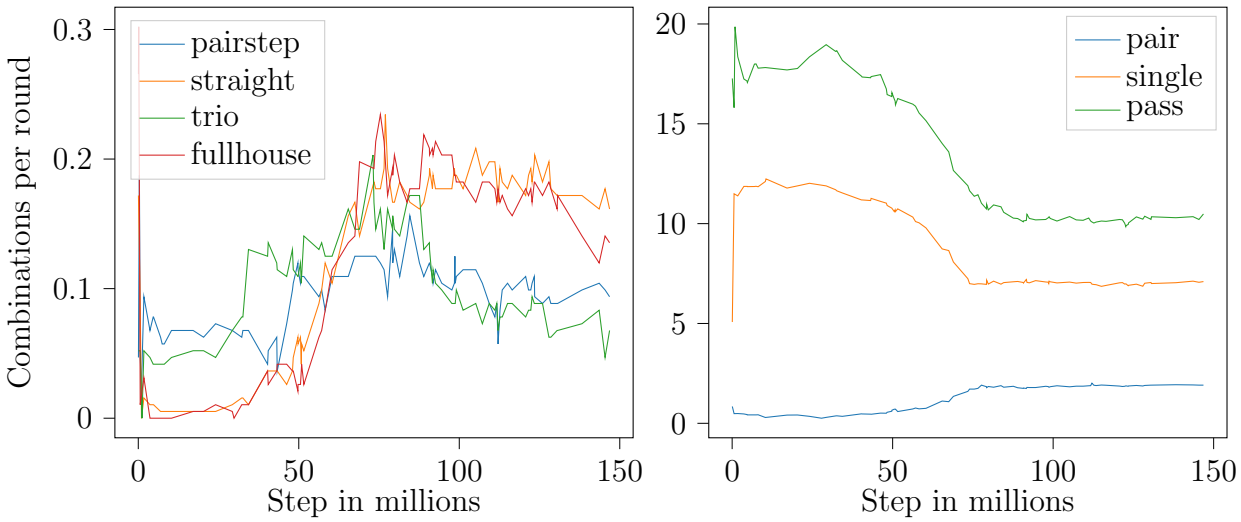
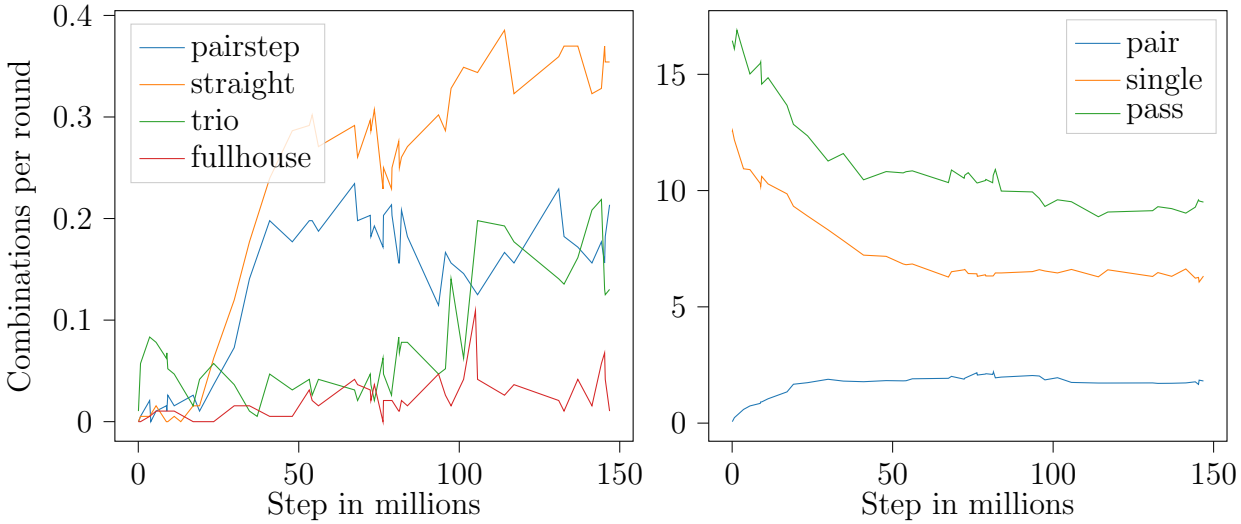


Figure A.3: Combination Frequency PPO



Bibliography

[1] Wolfgang Ertel. *Introduction to Artificial Intelligence, Second Edition*. Undergraduate Topics in Computer Science. Springer, 2017.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Process. Mag.*, 34(6):26–38, 2017.

[3] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, 1959.

[4] Yuxi Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.

[5] Quentin J. M. Huys, Anthony Cruickshank, and Peggy Seriès. *Reward-Based Learning, Model-Based and Model-Free*. Springer, 2014.

[6] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Mach. Learn.*, 8:279–292, 1992.

[7] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[9] Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *CoRR*, abs/1911.10635, 2019.

[10] Yuxi Li. Reinforcement learning applications. *CoRR*, abs/1908.06973, 2019.

[11] Peter Müller. Tichu bot. *DisCo semester Thesis, ETH Zurich*, 2020.

[12] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

[14] Burch, Neil. Time and Space: Why Imperfect Information Games are Hard.

[15] Matej Moravcík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael H. Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *CoRR*, abs/1701.01724, 2017.

[16] Donghwan Lee, Niao He, Parameswaran Kamalaruban, and Volkan Cevher. Optimization for reinforcement learning: From a single agent to cooperative agents. *IEEE Signal Process. Mag.*, 37(3):123–135, 2020.

[17] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[18] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for AI research. *Artif. Intell.*, 280:103216, 2020.

[19] Castelein, K.J.B. (Koen). *Determinization for Monte Carlo Tree Search n the Card Game Tichu*. Bachelor Thesis, Leiden University, 2016/2017.

[20] Martha Vlachou-Konchylaki and Stavros Vassos. Combining deliberation and reactive behavior for AI players in the mini-tichu card-game. In Georgios N. Yannakakis, Espen Aarseth, Kristine Jørgensen, and James C. Lester, editors, *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14-17, 2013*, pages 453–454. Society for the Advancement of the Science of Digital Games, 2013.

[21] A.G. Smith. Tichu game rules. `https://fatamorgana.ch/fatamorgana/tichu/english-rules`, 1993. Accessed: 2024-07-16.

[22] Lukas Pestalozzi. Tichu. `https://github.com/lukaspestalozzi/Tichu`, 2017. Accessed 2024-07-16.

[23] Laurent Sindayigaya. Machine learning algorithms: A review. *Information Systems Journal*, ISJ-RA-3392:6, aug 2022.

[24] Richard Bellman. Some applications of the theory of dynamic programming - A review. *Oper. Res.*, 2(3):275–288, 1954.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[26] Stable Baselines Team. Deep q-network (dqn), 2024. Accessed: 2024-08-08.

[27] Jie Tang and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 1000–1008. Curran Associates, Inc., 2010.

[28] Stable Baselines Team. Proximal policy optimization (ppo2), 2024. Accessed: 2024-08-08.

[29] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *CoRR*, abs/1805.01954, 2018.

[30] Stable Baselines Team. Pre-training (behavioral cloning), 2024. Accessed: 2024-08-08.

[31] Brettspielwelt. Tichu expert data from brettspielwelt, 2024. Accessed: 2024-07-08.

[32] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.