Using Q-Learning to train a 4×4 Connect Four Subgame Evaluator

Aarush Kumbhakern

March 29, 2025

1 Introduction

Connect Four is a two-player board game played on a 7-column by 6-row grid where players alternate dropping pieces, aiming to connect four of their pieces in a row. The game has significant strategic complexity: there are approximately 4.5 trillion possible (legal) board configurations in the standard 7×6 grid. Solving Connect Four (finding the optimal strategy) was a non-trivial accomplishment, achieved in 1988 [Wikipedia contributors, 2025], showing that with perfect play the first player can force a win.

Reinforcement learning (RL) is a paradigm in which an agent learns optimal behavior through trial-and-error interactions with an environment, guided by feedback in the form of rewards. A prominent algorithm in this paradigm is Q-learning, a model-free RL method that learns an action-value function $Q(s, a)$ representing the expected future reward of taking action a in state s. The agent updates its Q-values iteratively as it explores the state space, and under suitable conditions this process converges toward optimal values that define an optimal policy. In a tabular implementation of Q-learning, these values are stored in a Q-table - essentially a lookup table enumerating every possible state-action pair and its learned value. Tabular Q-learning is guaranteed to find an optimal policy in discrete environments given enough exploration. However, it is impractical for games with very large state spaces (like full-size Connect Four) because storing and exploring all possible states becomes infeasible. In such cases, more heuristic approaches or state space reductions are necessary.

One way to manage the complexity of Connect Four is to study a smaller version of the game. In this project, we focus on a 4×4 board subgame (4 rows and 4 columns) with the same connect-four objective. This yields only 16 board cells, which dramatically shrinks the state space (to up to 4.3 million legal states) and

makes near-exhaustive tabular learning more feasible. The idea is to train a Q-learning agent on this simplified 4×4 subgame so that it may effectively learn an evaluation function for local Connect Four positions, capturing fundamental patterns such as setting up "three-in-a-row" or blocking an opponent's threat, all in a tractable setting. We hypothesize that the agent will be able to learn these local tactics in the 4×4 environment and that such learned knowledge can be transferred to the full game.

After training the agent on the 4×4 subgame, the learned Q-table is applied to the standard game via a sliding-window approach. The full 7×6 board is scanned for all contiguous 4×4 regions (sub-board windows). The agent treats each such region as if it were a state in the smaller game, using its Q-table to evaluate potential moves within that window. These local evaluations are combined to inform the decision on the actual full board, and the agent selects the move that appears most promising across all windows. This sliding-window method leverages the assumption that good local patterns lead to good global outcomes, and is conceptually similar to how human-designed heuristics evaluate Connect Four boards by counting advantageous patterns - for example, favoring moves that create three-in-a-row or block the opponent's three-in-a-row.

This project aims to answer the following question: can a tabular Q-learning agent trained on a 4×4 Connect Four subgame provide an effective evaluation strategy for the full 7×6 game when applied through a sliding window, and thereby demonstrate that tabular reinforcement learning is viable in a dynamic, adversarial game environment like Connect Four under state space constraints?

2 Data

This project uses a custom-built implementation of a generalized Connect-X environment; data is generated as a part of the training loop via:

a) (optional/probabilistic) random board initialization

b) player-alternating random play

The Q-tables are indexed by tuples of `(player, state)` where player $\in \{1, 2\}$ is the current player, and state is a tuple of 16 cells with values $\{0, 1, 2\}$ for empty, player 1's piece, player 2's piece. The `player` element ensures that we align the perspective of the window state with the game's current player to move.

3 Methods

3.1 Tabular Q-Learning Algorithm

We employed a standard tabular Q-learning approach [Sutton and Barto, 2018] to train an evaluator for a 4×4 Connect Four subgame. In this framework, the agent maintains a Q-value $Q(s, a)$ for each state s (board configuration) and action a (a column move). The Q-table is updated iteratively as the agent plays games and receives rewards. An ϵ-greedy policy was used for exploration: at each decision state, with probability ϵ the agent selects a random valid move (exploration), and with probability $1 - \epsilon$ it selects the move with the highest current $Q(s, a)$ (exploitation). We initialized ϵ relatively high (favoring exploration) and decayed it towards a small floor value (e.g. $\epsilon_{\min} = 0.05$) over time to gradually shift from exploration to exploitation as learning progressed. The decay was adjusted periodically based on performance: if the agent's recent average reward improved, ϵ was geometrically decayed (to encourage exploitation of the better policy); if not, a slight ϵ boost was applied to reintroduce more exploration.

During training, Q-values were updated using the Q-learning update rule after each move. Given a transition from state s to s' via action a with received reward r, the update is:

$$Q_{\text{new}}(s, a) \leftarrow Q(s, a) + \alpha \Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \Big], \tag{1}$$

where α is the learning rate and γ is the discount factor. This update is applied after each action, treating each drop of a piece as a one-step transition. The term $\max_{a'} Q(s', a')$ is the highest Q-value for the next state s', reflecting the assumption that the agent (and in self-play, its opponent as well) will act optimally henceforth. In our implementation, we set $\gamma = 0.9$ to value future rewards, and a small α (e.g. 0.05) to ensure steady convergence.

3.1.1 Reward Shaping

The reward structure was designed to encourage winning and strategic play. The agent receives a reward of +1 for a win (i.e. if its move causes a Connect-4), 0 for a draw, and -1 for a loss. In addition, we incorporated a small block reward: if the agent's move successfully blocks an immediate (i.e. it places a piece that prevents the opponent from connecting 4 on their next turn) or partial (i.e. agent's move blocks a future setup) opponent threat, it receives a small positive reward (0.1 in our setup). This reward shaping provides an incentive for defensive moves that prevent losing threats, similar in spirit to reward-shaping techniques in reinforcement learning to encourage useful intermediate behavior. All rewards are delivered at the time of the move. Wins and losses terminate an episode (game), whereas block rewards may be obtained on non-terminal

moves.

3.2 Training Phases

Training was divided into three sequential phases, each aimed at progressively increasing the difficulty of the opponent and improving the quality of the learned Q-table. Throughout all phases, training consisted of many episodes of gameplay (on the order of 10^5 games per phase), and the Q-table was carried over (and further updated) from one phase to the next. The use of phased training ensured that the agent first learned basic gameplay against a naive opponent before facing more challenging self-play scenarios.

3.2.1 Phase 1: Training against a Random Opponent

In the first phase, the agent learned by playing against an opponent that selects moves uniformly at random. The Q-table was initialized (to zero for all state-action pairs) and updated through repeated games where one player is controlled by our learning agent and the other by the random policy. This scenario provides a gentle introduction to the game dynamics: the random opponent offers a broad and unbiased sampling of the state space, allowing the agent to experience varied board configurations. By the end of Phase 1, the agent's Q-table had learned to reliably achieve wins against random play and to avoid obvious losing moves.

Key outcomes of this phase included learning the value of immediate connect-4 wins (high positive Q-values for moves that complete a four-in-a-row) and the disvalue of moves leading to immediate losses (large negative Q-values for states where the opponent can win next).

3.2.2 Phase 2: Self-Play Training

In Phase 2, we leveraged pure self-play to improve the agent's strategy beyond the level of a random opponent. Self-play is a powerful technique in reinforcement learning for two-player zero-sum games, as demonstrated in classic and modern research [Tesauro, 1995, Silver et al., 2018]. We took the Q-table resulting from Phase 1 and continued training it by having the agent play against a copy of itself. In each self-play game, both players followed the same policy derived from the current Q-table (using ϵ-greedy moves for continued exploration). This means the agent was effectively playing both sides of the 4x4 board. To implement this, at the start of each game we designated one player as the learner and the other as the opponent for bookkeeping, both using the same Q-table for decision-making. The Q-learning updates were applied after every move (opponent and learner) from the perspective of the learning agent: e.g. if the move was made by the opponent copy and led to a win for the opponent, the reward was treated as a loss, -1, for the learning agent.

Self-play training forces the agent to play against increasingly stronger opposition as its own policy improves; any weakness in the strategy will eventually be exploited by its copy, creating pressure to correct it in the Q-table. Over many self-play games, the Q-values adjusted to reflect deeper strategic value, not just capitalizing on random mistakes. For example, the agent learned to set up future wins and block the opponent's potential connect-4 threats more consistently. Notably, self-play can lead to the emergence of sophisticated tactics with no external input, as the agent and its mirror gradually "teach" each other through their mistakes [Tesauro, 1995].

3.2.3 Phase 3: Adversarial Training with a Frozen Opponent

The final phase introduced an adversarial self-play regimen using a frozen opponent policy. The idea here is akin to iterative policy refinement: the agent plays against a fixed version of itself from the point in time immediatly before training, allowing it to improve by exploiting the fixed opponent's weaknesses. This improved agent becomes the new benchmark opponent for the next iteration.

In implementation, we took the Q-table from Phase 2 (self-play) and froze a copy of it to serve as the opponent's policy. We then continued training a fresh instance of the Q-table (initialized with the same Phase 2 values) by playing games where the opponent's moves were selected according to the frozen Q-table. Specifically, in each training game of this phase, one player uses the learning Q-table (updated via Q-learning as usual) and the other player uses the frozen Q-table (which is not updated within the game). After one generation was completed, we set the frozen Q-table equal to the newly learned Q-table - effectively, the agent's latest strategy becomes the opponent for the next round. This process was iterated for several generations (we used 5 generations in our experiments). Each generation thus sees the agent tackling a stronger version of itself, reminiscent of the training procedure used by Silver et al. (2018) in AlphaZero, where the network is periodically updated by self-play and each new network competes against the previous version.

After a large number of games (within a single generation), the learning Q-table typically surpasses the frozen one in performance, achieving a $\approx 600\%$ increase in win rate over a large number of episodes.

This adversarial training with a frozen opponent helps stabilize learning in the two-player setting by ensuring that during any batch of games, the opponent's strategy is stationary, allowing the Q-learning algorithm to converge toward a best response to that strategy. Then the strategy is updated, gradually approaching an equilibrium. The final Q-table after these generations constitutes the trained 4×4 subgame evaluator.

3.3 Sliding-Window Evaluator for the Full Game

With a Q-table trained for 4×4 Connect Four, our next step was to apply it to the standard Connect Four board (7 columns $\times$ 6 rows). We designed a sliding-window evaluator architecture that uses the 4×4 Q-table as a local heuristic evaluator across the larger board. The core idea is to break down the 7×6 board into smaller 4×4 sections (windows), evaluate each section using the learned subgame Q-values, and then combine those evaluations to inform a move decision on the full board.

3.3.1 4×4 Window Extraction from a 7×6 Board

The full Connect Four board is scanned for all possible contiguous 4-row by 4-column subgrids (windows), with each such 4×4 window seen as a mini Connect Four board.

There are 12 possible 4×4 subgames on a 7×6 board (since horizontally the window can start at columns 1 through 4, and vertically it can start at rows 1 through 3, in a 1-indexed sense). However, not all windows are equally relevant to the current game state due to the gravity rule (pieces stack from the bottom). We therefore extract windows in a gravity-aware manner: for each group of 4 adjacent columns, we consider the lowest 4×4 window that is not completely filled (unplayable). In practice, for each set of columns $[(c, c+1, c+2, c+3)]$ for $c = 1$ to 4, we start from the bottom of the board and find the highest 4-row segment that still contains empty cells (or reach the top of the board). That window becomes the representative subgame for that column group. This heuristic ensures that we focus on the portion of each column group where new moves can still be played or where the game action is concentrated, and always make decisions with full information (no false empty bottom rows).

As a result, at any given state we extract up to four 4×4 windows from the full board, each window capturing the local configuration of pieces in a region of the board that is relevant for potential moves. Each extracted window is represented as a state in the same format as used in the Q-learning phase. These window states serve as inputs to our learned Q-evaluator.

3.3.2 Q-Value Evaluation and Move Selection Across Windows

For each extracted 4×4 subgame window, we query the learned Q-table to evaluate the potential moves within that window. Specifically, let s_w denote the state of a window w (including the current player perspective). We retrieve the Q-values $Q(s_w, a)$ for $a = 1, \ldots, 4$ corresponding to dropping a piece in each column of that 4×4 subgame. These Q-values estimate the long-term value of making each move restricted to that subgame. Because the subgame is part of the larger board, we map each local action a (1-4) to the corresponding column

on the full 7-column board. For example, if the window w spans columns 3-6 of the full board and the Q-table suggests that action $a = 2$ (the second column of the window) has the highest value in w, this corresponds to column 4 of the full board as a candidate move. Thus, each window yields at most one recommended move (the valid action with the highest Q-value in that window) along with its Q-value. In effect, each local window "votes" for the best move as judged within that local context, and it provides a value (score) for how good that move is.

Once every window has been evaluated, we must combine their recommendations into a single decision for the full board. We experimented with different aggregation schemes for these local Q-value estimates. The approach we adopted was to select the move with the highest Q-value among all window evaluations - as each window was evaluated using the same Q-table, the confidence levels between the selected actions was baked into the Q-table itself. Formally, if the extracted windows are $w_1, w_2, \ldots, w_k$ (with $k \leq 4$) and each yields a best local action a_i^* (mapping to global column c_i^*) with value $Q_i^* = Q(s_{w_i}, a_i^*)$, we choose the move corresponding to $\arg \max_{i \in \{1,\ldots,k\}} Q_i^*$. This max-of-maxima strategy means the move selected is the one that some local window rates most highly. The rationale is that a single strongly favorable local configuration (such as a immediate win or a forced win setup) should dictate the global action.

In summary, the sliding-window evaluator works as follows: extract relevant 4×4 subgames from the 7×6 board, use the learned Q-table to evaluate the best move in each subgame, and then select the full-board move that has the highest local Q evaluation. This procedure allows the knowledge learned in the small 4×4 Connect Four to be applied to the standard game by treating the latter as a collection of overlapping smaller games. All game decisions in the full 7×6 Connect Four are made by this evaluation mechanism, effectively using the Q-table as a heuristic function guiding the agent's play on the full board.

4 Experimentation

One major issue encountered was that not all 4×4 board states (windows) that appear in a full 7×6 Connect Four game are reachable through alternating legal play confined to a 4×4 subgame. In a standard Connect Four game, gravity and turn-taking impose constraints on piece configurations. Certain local configurations of pieces in a 4×4 region might require sequences of moves that are impossible in the isolated subgame (e.g. a pattern that would require one player to play twice in a row). As a result, when the learned Q-table evaluator was applied as a sliding window on the full board, it sometimes encountered 4×4 states that were never seen during training. Given the deterministic nature of the Q-table window evaluator, these unseen states had no recorded Q-values, leading to lookup failures or default values. This gap highlighted a generaliza-

tion failure: the agent lacked the ability to estimate values for novel-but-similar states because of the limited coverage of the subgame state space.

4.1 Random Starts

To mitigate the state-space coverage problem, a random starts approach was used: a fraction of training episodes were initialized to random gravity-valid board configurations instead of an empty board. In practice, this meant that the 4×4 subgame would sometimes start with a few moves already played. These random start states force the Q-learning agent to confront board patterns that would otherwise be unreachable from an empty start under normal play. By doing so, the agent can acquire Q-values for a broader set of states. This technique is akin to the concept of exploring starts in reinforcement learning, which is known to improve state-space exploration coverage [Sutton and Barto, 2018]. Empirically, however, while the efficiency of the agent on a 4×4 game was not affected, we saw minimal improvement in the full 6×7 case.

4.2 Function Approximation

The fundamental limitation of a tabular method is its inability to generalize: each state is treated independently, and any state not encountered remains effectively invisible to the agent. To overcome this, one can replace or augment the Q-table with a function approximator, such as a deep neural network - in such a setup, the agent would use a neural network $Q_\theta(s, a)$ with parameters θ instead of a table, and which would learn to predict Q-values for a given board state by generalizing from patterns it has seen during training. In essence, the network might learn abstract features (such as "three in a row" or "imminent win for player 2") that apply to many states, allowing it to evaluate unseen states by similarity to seen ones.

Despite being beyond the immediate scope of this project, this forms a promising direction to improve the evaluator's generalization on the full game.

5 Final Results

5.1 4×4 Subgame

Training logs yielded clear evidence of progressive performance improvements and enhanced stability over time. During the initial phase against a random opponent, the agent's mean reward per episode began near zero, consistent with an untrained Q-learning agent playing a random policy. As the Q-table was incrementally updated, the average reward exhibited a marked upward trajectory. After 200,000 episodes against the random opponent, the moving average reward stabilized at a substantially positive value of approximately 0.7 per game.

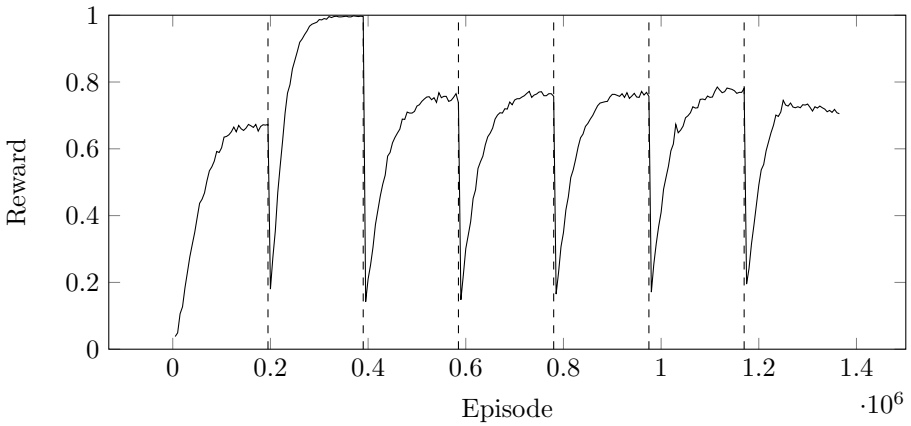

Figure 1: Agent avg. reward time-series data

This outcome demonstrates that the tabular Q-learning algorithm successfully acquired tactics sufficient to outperform a random policy.

In the subsequent self-play phase, a transient drop in average reward (to approximately 1.5) was observed. This decrease likely reflects the agent's initial struggle to outperform an equally capable version of itself, with a high frequency of draws or evenly matched wins and losses. As training progressed, however, the Q-table adapted further, surpassing the performance level attained during the random-opponent phase. A pronounced reduction in losses suggested that self-play encouraged the agent to prioritize avoiding defeat - ensuring a draw in situations where a win was uncertain - thereby demonstrating a more conservative and defense-oriented style of play.

At the outset of each generation in the adversarial phase, performance dips for the same reasons described for the self-play phase. Nonetheless, with continued training, the agent regained and then exceeded its prior performance, culminating in increasingly robust Q-table parameters. By the fifth generation, the learning curve plateaued (Figure 1), suggesting convergence toward an optimal or near-optimal policy for the subgame. Final evaluations showed that, against a random opponent, the agent achieved a nearly 75% win rate, and approximately 3% losses and 23% draws (Table 1).

The overall performance trend is illustrated in Table 1, where each training phase (random, self-play, and adversarial) yields incremental improvements in the agent's win rate and corresponding reductions in losses. Notably, the draw rate increases pas the random opponent phase, indicating that the agent adopted risk-averse tactics to prevent defeat whenever a win was not guaranteed. This escalation in overall stability, marked by high win rates and negligible loss rates,

underscores the efficacy of the adversarial training regimen in rectifying residual weaknesses exploitable by a competent opponent.

In conclusion, the combined training protocol produced a Q-learning-based evaluator that demonstrates strong tactical proficiency within its domain. The final Q-table successfully prioritizes winning moves, blocks threats from the opposing player, and secures draws from positions of disadvantage. Both the learning curves and test metrics confirm the value of the progressive training methodology, whereby each phase, ranging from random-opponent exposure to self-play and adversarial refinement, augmented the agent's strategic capabilities.

Training Phase	Win Rate	Loss Rate	Draw Rate
After Random Opponent	0.709	0.239	0.052
After Self-Play	0.691	0.056	0.253
After Adversarial (Final)	0.739	0.028	0.233

Table 1: Agent performance (fraction of games) against a random opponent after each training phase.

5.2 6×7 Full Game

Heuristic testing of the static-evaluator windowed agent served to prove the concerns regarding applying the Q-table agent to the larger game correct. As player 2, the agent was at the very least able to "chase" the user's moves around the board, but as player 1, the subagent's opinionated policy very quickly led to unevaluatable windows, and a complete loss of strategy in these areas.

6 Conclusion

Training a tabular Q-learning agent on a 4×4 Connect Four subgame produced a capable local evaluator, but using this evaluator on the full game had significant limitations. The agent successfully learned and converged to a sound policy in the subgame, and demonstrated good short-range tactical awareness. In some cases, the agent was able to play optimally "between" windows, and the sliding window approach in general showed a good deal of promise.

However, a significant limitation became apparent when deploying the subgame-trained agent on the full 7×6 game - the issue of unseen states due to the agent's deterministic nature once the Q-table is learned. If the table was incomplete or slightly suboptimal, the agent had no fallback, suggesting future imrpovements in the direction of deep Q-learning methods.

References

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition.

[Tesauro, 1995] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

[Wikipedia contributors, 2025] Wikipedia contributors (2025). Connect four — wikipedia, the free encyclopedia. Accessed: 2025-04-05.