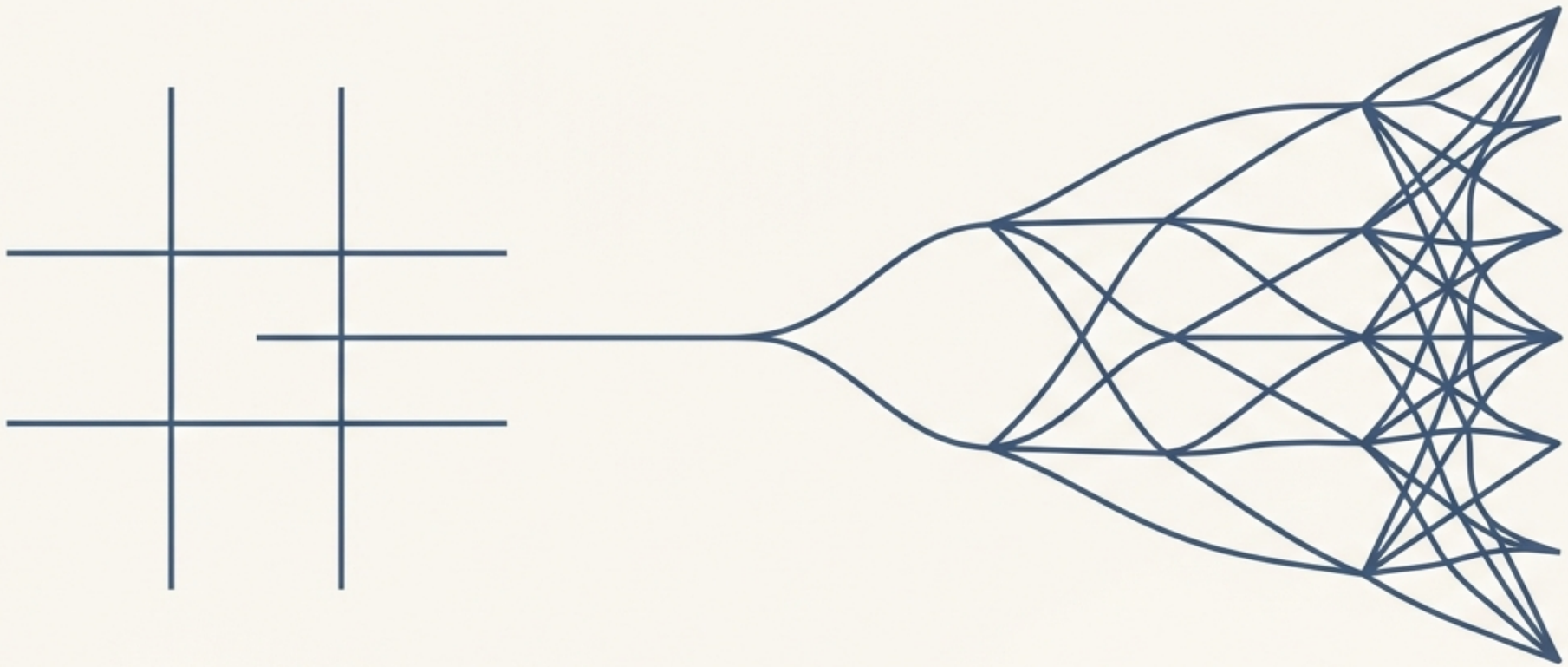
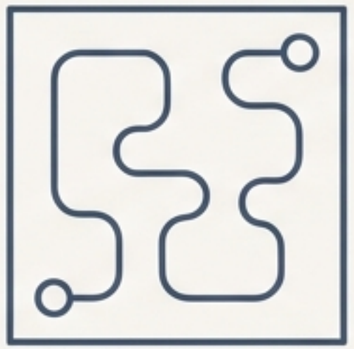
An Experimental Journey into Reinforcement Learning for Tic-Tac-Toe

From Tabular Methods to Neural Networks: A Comparative Analysis of Agent Performance



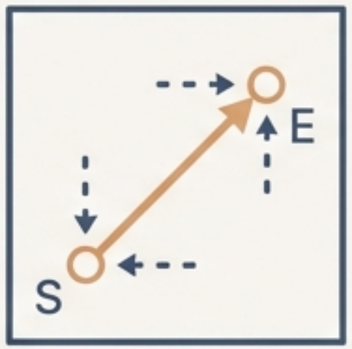
The Experimental Framework

This analysis distils the key findings from a series of experiments designed to train a Tic-Tac-Toe agent. We will conduct a comparative deep dive into three distinct Reinforcement Learning algorithms, examining the impact of hyperparamiter tuning on their performance and behaviour.



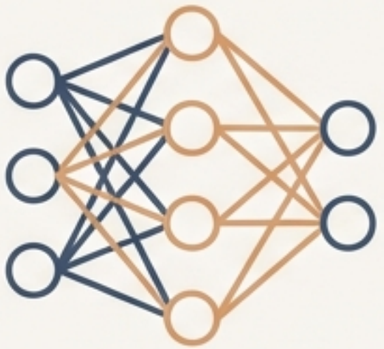
1. Monte Carlo

Learning from complete game outcomes.



2. Q-Learning

Learning from individual step-by-step actions.



3. DQN

Generalising learning with a neural network.

Chapter 01: Monte Carlo

Learning from Complete Episodes

The Impact of Learning Rate on Convergence

Observation

run	learning_rate	discount_factor	episodes_trained
fresh-cosmos-1	0.001	0.99	~55,000
leafy-bird-2	0.0001	0.99	~100,000

Key Finding: The higher learning rate (`0.001`) reached its performance plateau significantly faster, triggering early stopping around 55k episodes. The lower rate (`0.0001`) trained for the full duration, indicating slower convergence.

Interpretation

- Increasing the learning rate successfully accelerated learning, reducing the time required to reach a stable policy.
- While a learning rate that is too high can introduce instability, the `0.001` rate primarily improved time-to-plateau in this context without observable negative side effects.

The Agent's Horizon: Tuning the Discount Factor

Observation

run	learning_rate	discount_factor	episodes_trained	Q-Table Size
quiet-glade-3	0.0001	0.05	~53,000	Smaller
leafy-bird-2	0.0001	0.99	~100,000	Larger (~4000 states)

Key Finding: The high-discount agent (`0.99`) trained longer and explored a larger state space. The low-discount agent (`0.05`) converged quickly on a smaller Q-table.

Interpretation

- Tic-Tac-Toe outcomes are delayed; rewards often come only at the end of the game.
- A higher discount factor correctly encourages the agent to value long-term outcomes, enabling it to learn more complex strategies like blocks and forks. This comes at the cost of increased training time.

Chapter 02: Q-Learning

Learning Step-by-Step

Q-Learning Performance Snapshot

run	discount_factor	final_win_rate	final_loss_rate	final_tie_rate
comfy-pyramid-12	0.9	**0.80**	0.10	0.10
clear-frost-14	0.9	**0.79**	0.12	0.09
lilac-water-8	0.9	**0.78**	0.16	0.06
ancient-wind-10	0.5	0.71	0.09	0.20
exalted-smoke-9	0.5	0.57	0.16	0.27

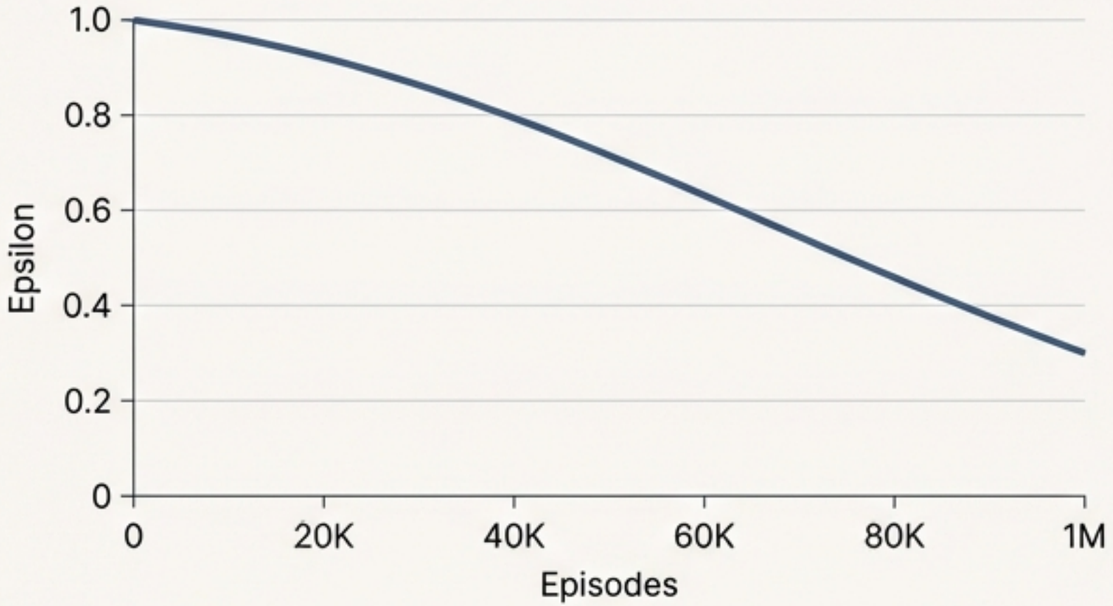
Key Insight

The configurations using a high discount factor (`0.9`) consistently achieved the highest final win rates, outperforming the lower-discount (`0.5`) group.

The Exploration vs. Exploitation Trade-off

All runs began with `epsilon=1` (full exploration). The `epsilon_decay` rate determined how quickly the agent shifted to an exploitation-focused policy.

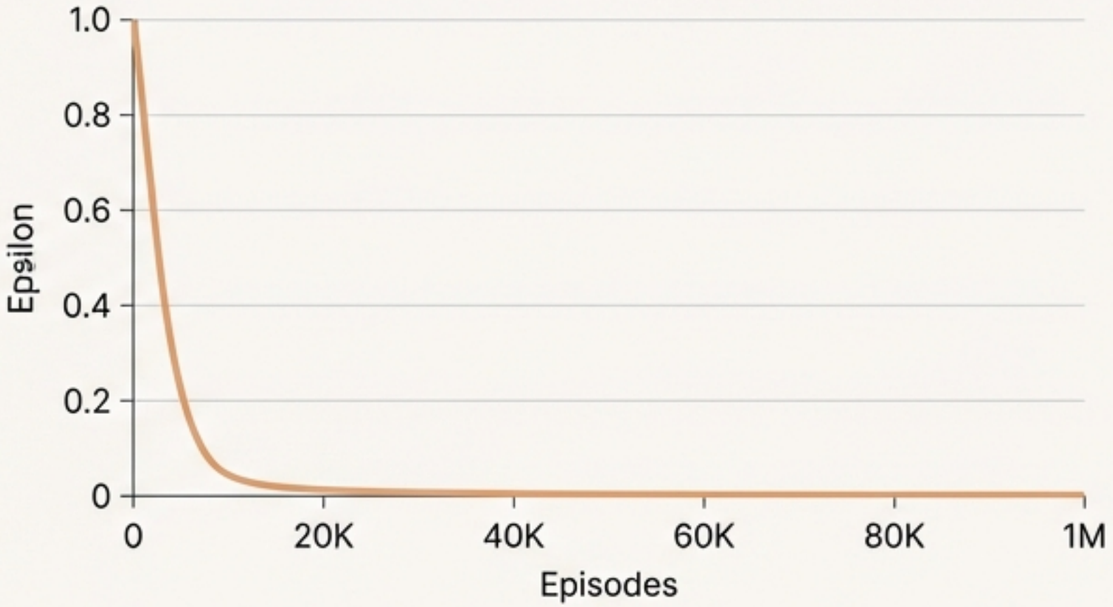
Slow Decay (e.g., `0.99999`)



Behaviour: Epsilon decreases gradually over the full 1 million episodes. Exploration is maintained for longer.

Impact: This can help the agent discover more optimal policies by avoiding premature convergence, but may inject destabilising randomness late in training.

Fast Decay (e.g., `0.995`)



Behaviour: Epsilon drops to its minimum value very early in the training process.

Impact: The agent locks into an exploitation strategy sooner. This can stabilise behaviour quickly but risks trapping the agent in a suboptimal policy it discovered early on.

Analysis of Theoralysis of Theoretical Scenarios

What if `learning_rate` was higher (e.g., 0.1)?

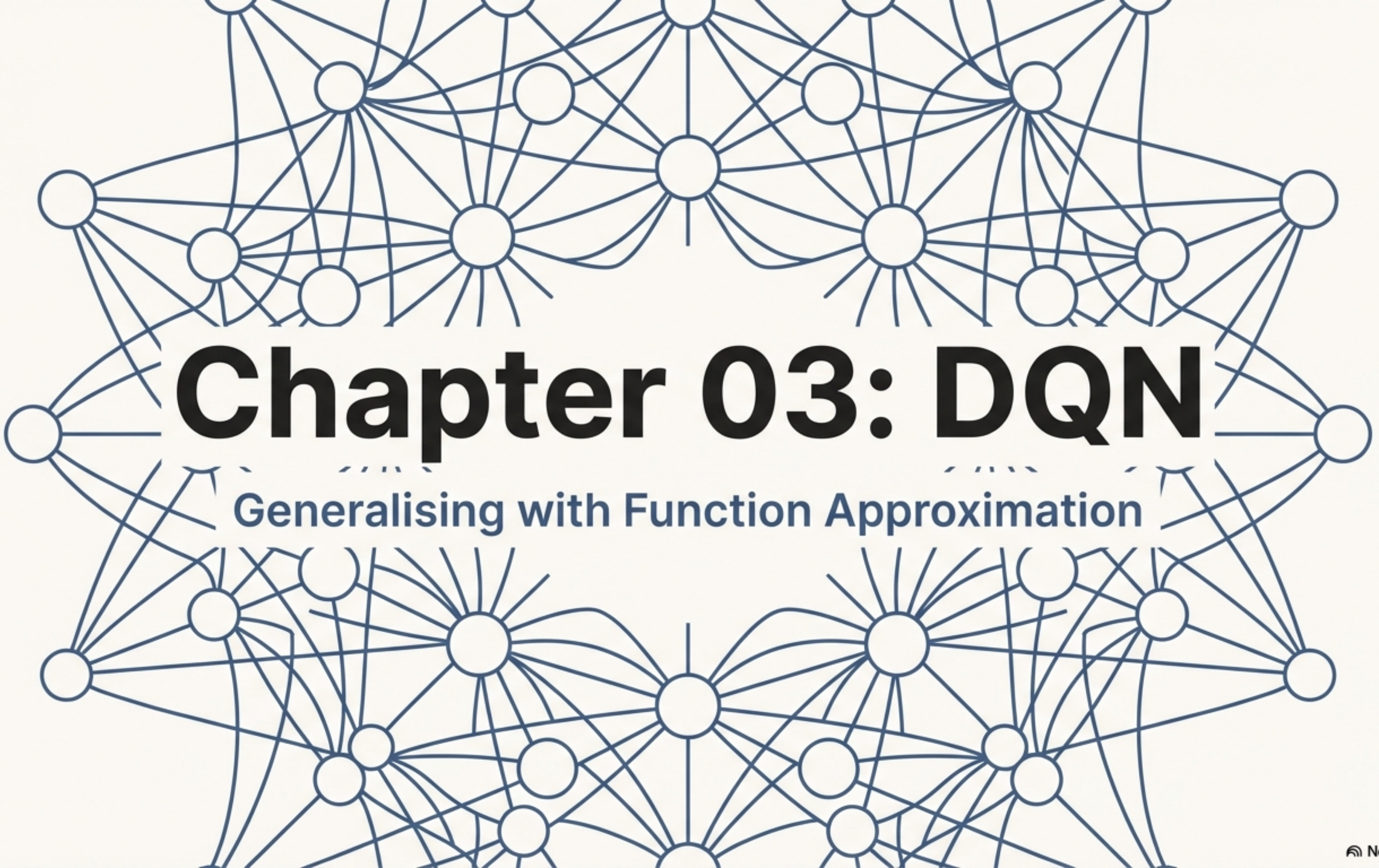
Expected Impact: Faster initial learning, but a significant risk of instability. High learning rates can cause larger, more erratic Q-value updates, potentially leading to the kind of late-training performance collapse seen in the `clear-frost-14` run.

What if `tie_reward` was negative?

Expected Impact: The agent would learn to treat draws as a penalty. This would likely cause it to pursue riskier lines of play to force a win, even when a draw is the optimal safe move. The result would be a lower tie rate, but a higher loss rate against a competent opponent.

What if we trained for 10M episodes?

Expected Impact: Likely diminishing returns. If a policy has already plateaued or become unstable (as seen in `clear-frost-14`), more episodes are not guaranteed to improve performance and could exacerbate overfitting or instability.

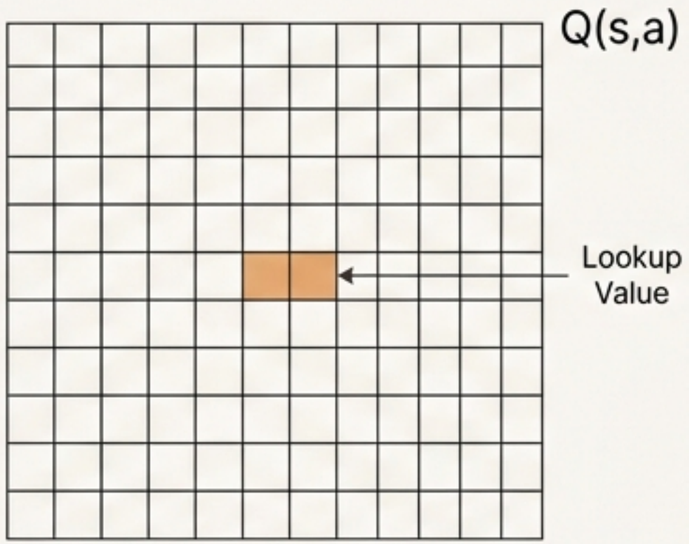


Chapter 03: DQN

Generalising with Function Approximation

Replacing the Q-Table with a Neural Network

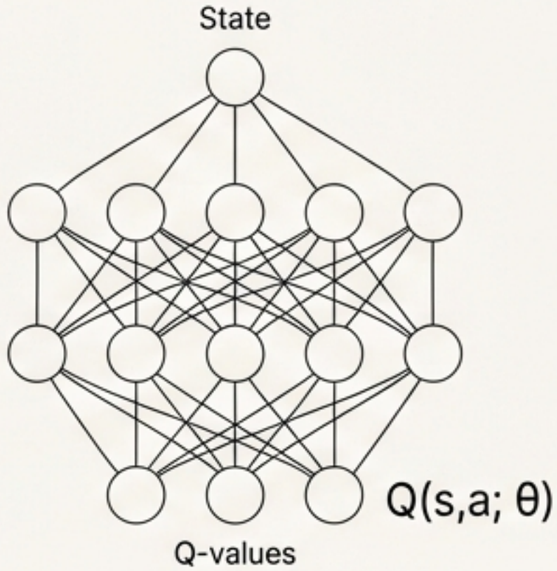
Classic Q-Learning



Stores Q-values for every possible state-action pair in a large lookup table: `Q(s,a)`.

Limitation: Does not scale to large or continuous state spaces. Cannot generalise learning between similar states.

Deep Q-Network (DQN)



Approximates the Q-function with a neural network: `Q(s,a; θ)`.

Mechanism: The network takes the game state as input and outputs a vector of Q-values for all possible actions.

Key Innovations for Stability:
Experience Replay: Stores transitions in a `replay_memory` to break temporal correlations.
Target Network: Uses a separate, slowly-updated `target_model` to provide stable Bellman targets during training.

A Proposed DQN Architecture for Tic-Tac-Toe

Input State Representation

A flattened 9-element vector representing the board.

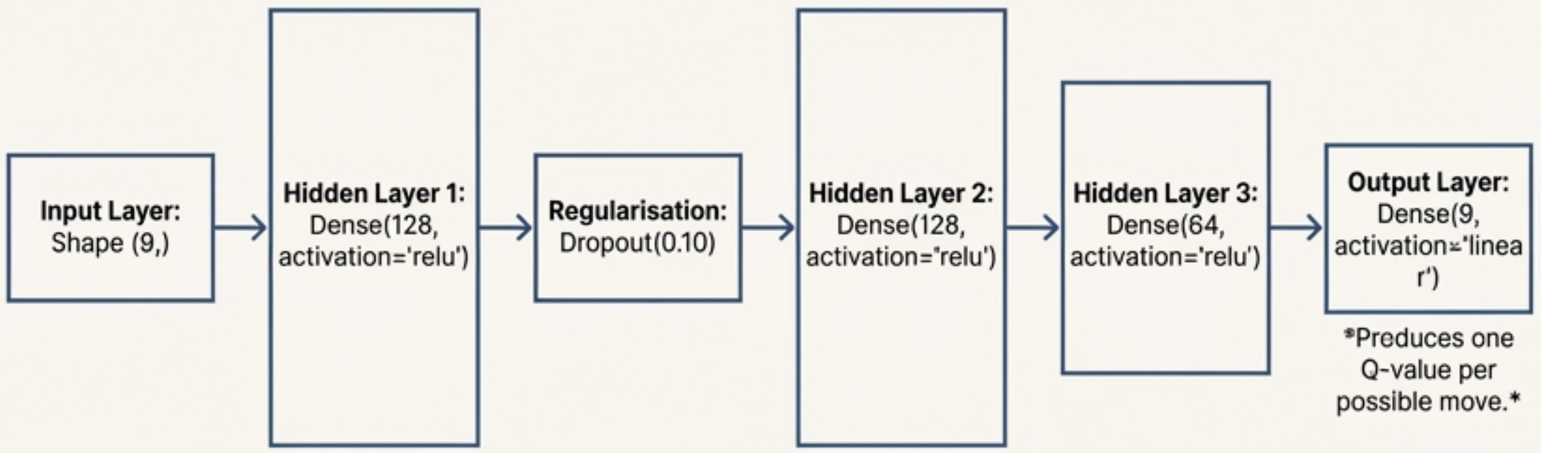
Encoding: `X = 1`, `O = -1`, `Empty = 0`.

Training Details

Loss Function: Huber (more robust to outliers than MSE).

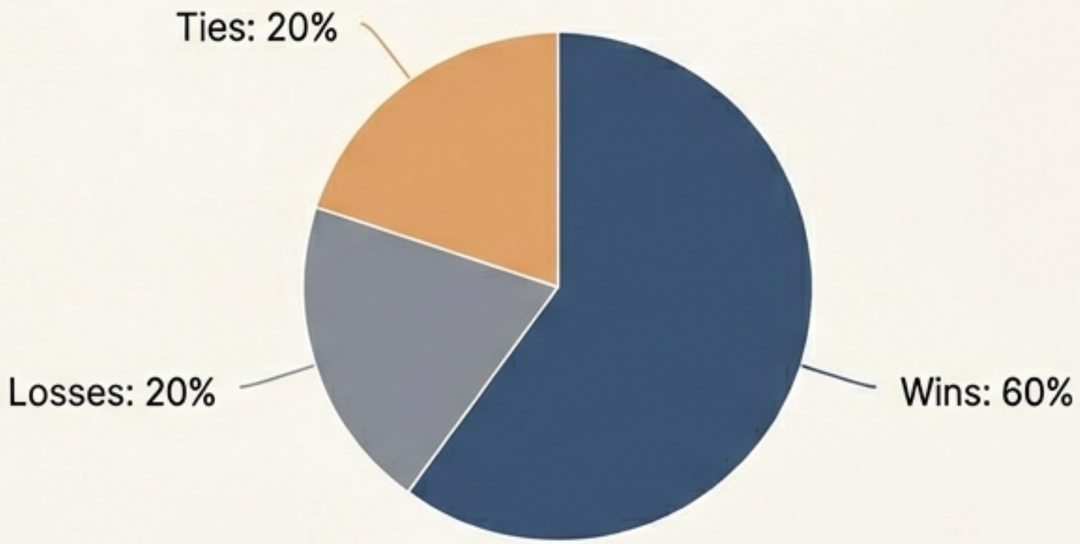
Optimiser: Adam.

Model Architecture (Multi-Layer Perceptron)



Observed DQN Performance and Behaviour

Game Outcome Distribution



The agent learned a winning policy, securing victory in a clear majority of evaluation games.

Behavioural Analysis

- **Actions Per Episode**: Plots indicate games are frequently short, with the agent often taking 5 actions. This is consistent with the agent learning to force decisive outcomes (wins or losses).
- **Interpretation**: The significant percentage of losses and ties suggests the learned policy is effective but not yet fully optimal. Against a perfect opponent, optimal Tic-Tac-Toe play converges towards a much higher proportion of draws.

Synthesis: A Comparative View of the Three Approaches

Feature	Monte Carlo (MC)	Q-Learning	Deep Q-Network (DQN)
Core Mechanism	Learns from complete episode returns. Updates are applied only at the end of a game.	Learns from individual state-action transitions (bootstrapping). Updates after each step.	Approximates the Q-function using a neural network. Learns from batches of past experience.
Key Advantage	Unbiased value estimates as it uses true, full returns. Simple to implement.	More sample-efficient than MC as it learns from every step.	Generalises across similar states. Can handle vast or continuous state spaces.
Primary Limitation	Inefficient for long episodes as credit assignment is delayed. High variance.	Estimates are biased due to bootstrapping. Struggles with very large, discrete state spaces.	Prone to training instability. More complex hyperparameters to tune (network architecture, replay buffer size, etc.).
Scalability	Poor. Requires visiting every state many times. Not feasible beyond small problems.	Moderate. Limited by memory required to store the Q-table.	High. The definitive approach for complex problems where tabular methods are impossible.

From Look-up Tables to Generalised Strategy

The journey from Monte Carlo to DQN is a clear illustration of the core trade-off in applied Reinforcement Learning: the balance between the sample efficiency and stability of tabular methods and the powerful generalisation and scalability of function approximation. For a simple, constrained environment like Tic-Tac-Toe, all three methods can find a solution. However, their underlying mechanics reveal a fundamental progression in how an agent can learn to master its environment.