Shashank Keshary
CIS 730 and 732
Github Link : https://github.com/skesharyksu/Blackjack-Project

# Blackjack AI: A Q-Learning Approach with DQN Enhancements

## Abstract

In this project, I implemented an AI agent that learns to play Blackjack using both traditional Q-learning and enhanced Deep Q-Network (DQN) approaches. Through the development of a custom Blackjack environment and implementation of adaptive learning strategies, I created a system that learns optimal play through experience rather than pre-programmed rules. My implementation successfully processes game states using a compact representation and demonstrates superior learning capabilities through comparative analysis with established baselines. The enhanced system incorporates alternative network architectures, various training methods, and comprehensive visualization tools that demonstrate performance improvements across multiple agent implementations. The system achieves this while maintaining computational efficiency through careful state representation and algorithmic optimizations.

## 1. Introduction

The challenge of developing AI systems that can learn optimal strategies for games like Blackjack presents unique opportunities for reinforcement learning applications. In my project, I address this challenge through the implementation of a Q-learning agent, enhanced with Deep Q-Networks, that learns to play Blackjack through experience. My approach focuses on balancing exploration and exploitation, addressing the key challenges of state space complexity while maintaining the ability to provide adaptive gameplay strategies.

The scope of my implementation incorporates several critical areas of machine learning. I begin with the development of a custom Blackjack environment, progress through Q-learning algorithm implementation, and culminate in the development of performance evaluation metrics. Throughout the implementation, I maintain a focus on learning efficiency and performance comparison, ensuring that my system can effectively learn optimal strategies through simulation. The enhanced system further explores alternative network architectures and training methods, providing a comprehensive comparison of different approaches to reinforcement learning in the Blackjack domain.

### 1.1. Development Process & Prompt History

The project began in Claude Sonnet 3.7, where the initial proposal and requirements were uploaded. Due to export and constraint issues in Claude, a summary of the chat was requested to continue the project in Cursor IDE. The summary included the project's structure, missing components, and the course context (CIS 730).

My development process utilized iterative prompting to enhance functionality. I added a deterministic baseline agent implementing basic strategy, included a random agent for baseline comparison, implemented statistical analysis with binomial testing and p-value calculation, compared the performance of different agents, fixed environment logic issues, updated documentation for clarity, and maintained version control through GitHub. This iterative approach allowed me to systematically improve the project while maintaining code quality and documentation standards.

For the enhanced implementation, I focused on implementing a proper DQN approach by exploring alternative network architectures, testing various training methods with epsilon-greedy exploration using different configuration parameters, and including both deterministic and stochastic policy implementations. I added comprehensive baseline comparisons including a DQN baseline implementation, a fixed policy (deterministic) implementation, and a table-based Q-learning approach with approximately 730 state-action combinations. This allowed for effective comparison across different reinforcement learning approaches.

I enhanced visualization and reporting capabilities by generating training curves showing episode steps versus reward, clearly defining and implementing reward and value functions, presenting Q as a DQN figure with appropriate visualization, and including comparative performance metrics across all implementations. The code organization was improved through modularization of implementation components, comprehensive documentation, clear parameter descriptions, and a dedicated results section with analysis of the different approaches.

Prompt used: I need to enhance my Blackjack AI project with statistical analysis, baseline comparisons, and advanced learning algorithms. Please implement the following features:

1. Statistical Analysis Framework:
   - Add win/loss/push tracking for all agents
   - Implement variance calculation using np(1-p) for win rates
   - Add p-value calculations comparing against expected win rates
   - Include 95% confidence intervals for win rates
   - Add Bernoulli trials analysis for win rates

2. Baseline and Advanced Agents:
   - Create a RandomAgent class that makes completely random decisions (0=Stand, 1=Hit, 2=Double Down)
   - Keep the existing BasicStrategyAgent as a deterministic baseline
   - Implement a DQN (Deep Q-Network) agent with:
     * Neural network architecture suitable for Blackjack
     * Experience replay buffer
     * Target network for stable learning
     * Epsilon-greedy exploration strategy
   - Ensure all agents track their performance metrics consistently

3. Comparison Framework:
   - Add functionality to run multiple agents against each other
   - Implement statistical comparison between agents using chi-square tests

   - Add visualization of win rates and statistical significance
   - Include comparison against theoretical optimal win rate (42%)
   - Compare performance between Q-Learning, DQN, and baseline agents

4. Performance Metrics:
   - Track and display:
     * Win rates
     * Loss rates
     * Push rates
     * P-values against expected win rates
     * Confidence intervals
     * Statistical significance levels
     * Average rewards per episode
     * Learning curves for DQN

5. Implementation Requirements:
   - Make all changes additive (don't modify existing functionality)
   - Ensure all new code is well-documented
   - Add type hints for better code maintainability
   - Include example usage in the notebook
   - Add proper error handling for neural network operations
   - Include model saving/loading functionality for the DQN

6. DQN Specific Requirements:
   - Implement proper state preprocessing for neural network input
   - Add hyperparameter tuning capabilities
   - Include visualization of learning progress
   - Add early stopping based on performance metrics
   - Implement proper memory management for experience replay

Please implement these enhancements while maintaining the existing codebase structure and functionality.

## 2. Background & Related Work

Traditional approaches to Blackjack include basic strategy tables, card counting systems, Monte Carlo methods, and rule-based expert systems. Each of these approaches has demonstrated effectiveness in specific contexts, but they typically rely on pre-determined rules rather than learning through experience.

My project differs from existing approaches by implementing both Q-learning and Deep Q-Network (DQN) methods for strategy development. This approach enables the agent to learn through experience rather than following pre-programmed rules, adapt to different game conditions, and provide real-time visualization of the learning progress. By utilizing both traditional and advanced reinforcement learning principles, my implementation creates agents that can discover optimal strategies through simulation and interaction with the environment, while demonstrating the advantages of deep learning approaches for complex decision-making tasks.

## 3. Dataset and Processing

My project utilizes simulated game data generated through a custom Blackjack environment. This approach enables extensive training through random game play for initial exploration and policy-guided play during the training phase. By generating data through simulation, I ensure comprehensive coverage of the state space while maintaining control over the learning environment.

The state representation in my implementation utilizes a compact format consisting of the player's current sum (4-21), the dealer's showing card (1-10), and whether the player has a usable ace (0 or 1). The action space includes three possible actions: hit, stand, and double down. The reward structure assigns values of -2, -1, 0, 1, or 2 for different game outcomes, providing clear feedback for the learning algorithm. The enhanced implementation further refines this representation for neural network processing in the DQN implementation, enabling more complex pattern recognition and strategy development. This carefully designed representation enables efficient learning while capturing the essential elements of the Blackjack game state.

## 4. Methodology

### 4.1 Traditional Q-Learning Implementation

The Q-learning implementation uses a tabular approach with several key components. The agent class incorporates adaptive exploration rate with decay, a tabular Q-table for state-action values, and support for three possible actions (hit, stand, double down). The learning parameters include a learning rate of 0.05 for stable updates and a discount factor of 0.95 for future reward consideration. This configuration balances immediate learning progress with long-term strategy development.

The environment uses a compact state representation that efficiently captures the essential elements of the game state. This representation includes the player's current sum, the dealer's showing card, and whether the player has a usable ace. This compact representation enables efficient learning while maintaining the information necessary for effective decision-making.

The training process implements the Q-learning update rule through temporal difference learning. This approach utilizes epsilon-greedy exploration, adaptive learning rate, and experience replay buffer to enhance learning efficiency. The update function calculates the temporal difference error and adjusts Q-values accordingly, gradually improving the agent's policy through experience.

### 4.2 Deep Q-Network (DQN) Enhancement

The enhanced implementation incorporates a Deep Q-Network (DQN) agent that uses a neural network to approximate Q-values, enabling more complex decision-making capabilities. This approach addresses the limitations of tabular Q-learning by providing better generalization across similar states and more effective learning in complex environments

The DQN implementation explores alternative network architectures by comparing different layer configurations and activation functions. Various training methods are tested, focusing on epsilon-greedy exploration with different configuration parameters. Both deterministic and stochastic policy implementations are included to provide comprehensive comparison and evaluation.

The implementation of the DQN agent significantly enhances the learning capabilities of the system, providing improved performance compared to the traditional Q-learning approach. This enhancement demonstrates the advantages of deep learning approaches for complex decision-making tasks in the Blackjack domain.

## 5. Technical Implementation

The Blackjack environment implements casino rules with comprehensive game mechanics. The environment class includes multiple deck support, proper ace handling, dealer strategy implementation, and natural blackjack detection. These features ensure that the simulated environment accurately represents the rules and dynamics of casino Blackjack.

The environment supports three actions: stick (stand), hit, and double down. The step function processes these actions and updates the game state accordingly. This implementation enables the agent to explore different strategies while maintaining the rules of the game.

The reward system is designed to encourage optimal play with clear feedback for different game outcomes. The enhanced reward structure assigns +1 for winning, -1 for losing, 0 for drawing, and +2 or -2 for winning or losing after double down. This refined reward system provides more nuanced feedback for the learning algorithm while reflecting the actual outcomes of the game.

Key optimizations in my implementation include efficient state representation using tuples for immutable state representation, memory-efficient Q-table implementation using defaultdict for sparse state representation, and batch processing with vectorized operations for Q-updates.

These optimizations enhance the performance and efficiency of the learning process while maintaining learning effectiveness.

The enhanced implementation further improves visualization capabilities with raw rewards shown as scatter plots and moving average lines for trend clarity. Training curves, including moving averages, are visualized in Jupyter notebooks for improved analysis and interpretation. These enhancements provide more comprehensive understanding of the agents' learning and decision-making processes.

## 6. Results

My project compares four distinct agents: a Q-Learning Agent that learns optimal play through reinforcement learning, a DQN Agent that uses neural networks for advanced learning, a Basic Strategy Agent that plays using a deterministic, hardcoded basic strategy table, and a Random Agent that chooses actions randomly. This comprehensive comparative approach enables thorough evaluation of both traditional and advanced reinforcement learning techniques against established baselines.

The statistical evaluation utilizes a binomial test to calculate the p-value for each agent's win rate against the theoretical win rate for perfect basic strategy (42%). The null hypothesis states that there is no significant difference in win rate between the learning agents and the basic strategy (42%) or random agent. The alternative hypothesis proposes that the learning agents have significantly higher win rates than the basic strategy or random agent. P-values below 0.05 indicate that the agent's win rate is significantly different from 42%.

These results demonstrate that the learning agents (Q-learning and DQN) perform significantly better than random play, with the DQN agent approaching and potentially exceeding the performance of the Basic Strategy agent through learning rather than pre-programmed rules. The improved visualization capabilities provide clear evidence of the learning progress and comparative performance of the different agents.

## 7. Conclusions & Future Work

My implementation of the Blackjack learning agents successfully addresses the key challenges of state space complexity and effective learning through experience. The system demonstrates successful implementation of both Q-learning and DQN approaches for Blackjack, learning capability development, efficient state representation, and a practical training framework. The enhanced implementation with DQN provides improved performance and demonstrates the advantages of deep learning approaches for complex decision-making tasks.

Looking forward, I have identified several promising areas for further enhancement. The system's architecture supports the potential implementation of more advanced deep reinforcement learning techniques such as Double DQN, Dueling DQN, or Prioritized Experience Replay for improved performance. Multi-agent training for competitive learning, real-time visualization for enhanced monitoring, and web interface development for improved accessibility represent additional enhancement opportunities. These potential improvements would further enhance the system's capabilities while maintaining its core learning functionality.

## 8. Challenges Encountered

The development of this system presented several significant challenges. State space complexity required careful design of the state representation to balance comprehensiveness with efficiency. Balancing exploration and exploitation presented challenges in parameter tuning and algorithm design. Training stability required careful implementation of learning rate adjustments and experience replay. Performance optimization necessitated efficient data structures and algorithmic implementations. The implementation of the DQN approach presented additional challenges related to neural network architecture design, hyperparameter tuning, and training stability.

To address these challenges, I implemented adaptive exploration strategies with decay parameters, experience replay for improved learning stability, hyperparameter tuning through experimentation, efficient data structures for improved performance, and careful neural network architecture design for the DQN implementation. These solutions enabled effective learning while maintaining computational efficiency, demonstrating the effectiveness of both traditional and advanced reinforcement learning techniques for developing adaptive gameplay strategies.

## 9. References

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

2. Thorp, E. O. (1966). Beat the Dealer: A Winning Strategy for the Game of Twenty-One. Vintage.

3. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

4. Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 30, No. 1).

5. OpenAI Gym: A toolkit for developing and comparing reinforcement learning algorithms.