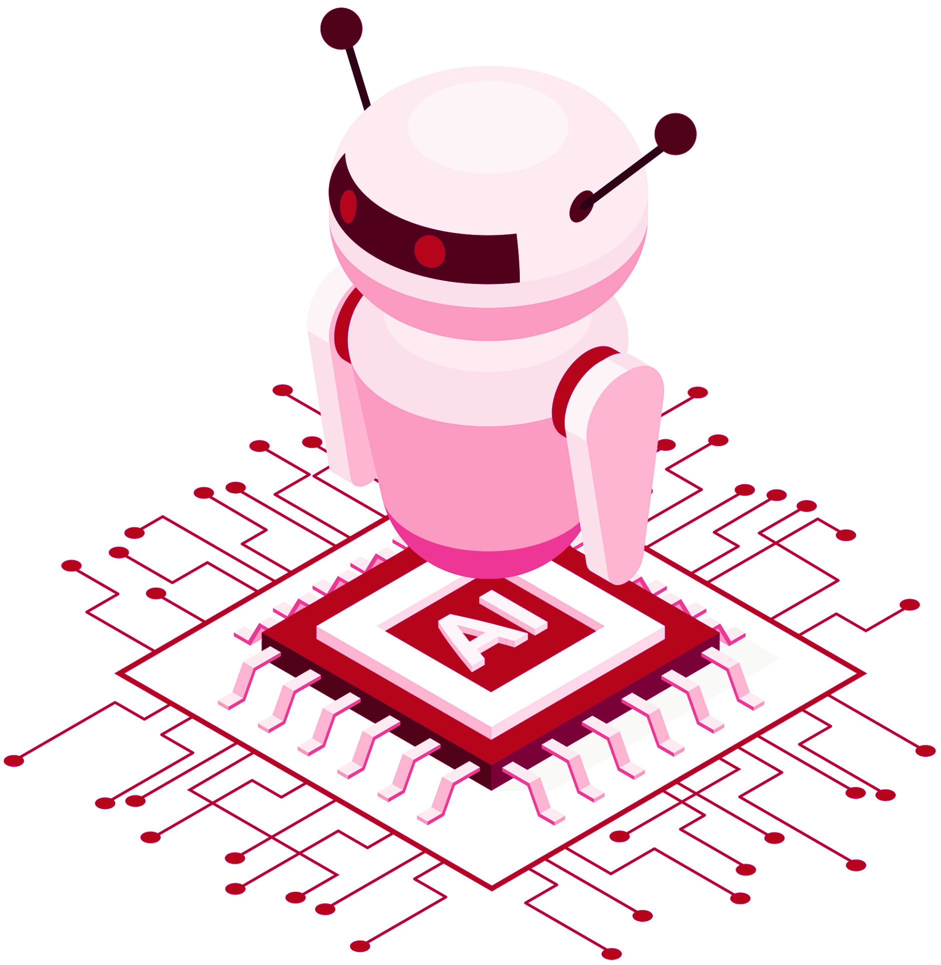

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 2:

Value-Based Methods

By:

Mohammad Taha Majlesi

810101504

Spring 2025 w

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	Points
Task 1: Epsilon Greedy & N-step Sarsa/Q-learning	40
Jupyter Notebook	25
Analysis and Deduction	15
Task 2: DQN vs. DDQN	50
Jupyter Notebook	30
Analysis and Deduction	20
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Epsilon Greedy

1.1 Epsilon 0.1 initially has a high regret rate but decreases quickly. Why is that? [2.5-points]

Epsilon 0.1 initially has a high regret rate because the agent starts with limited knowledge of the environment and must explore to discover the optimal policy. However, it decreases quickly because:

- **Low exploration rate**: With $\epsilon = 0.1$, the agent exploits 90% of the time, allowing it to quickly converge to a good policy once it discovers promising actions.
- **Fast learning**: The CliffWalking environment has deterministic dynamics, so the agent can quickly learn the consequences of actions and identify the optimal path.
- **Balanced exploration-exploitation**: The 10% exploration rate is sufficient to discover the optimal path without excessive random actions that would increase regret.

The rapid decrease in regret indicates that epsilon 0.1 provides an effective balance between exploration and exploitation for this environment.

1.2 Both epsilon 0.1 and 0.5 show jumps. What is the reason for this? [2.5-points]

The jumps in regret for both epsilon 0.1 and 0.5 are caused by:

- **Cliff encounters**: When the agent explores and falls off the cliff (receiving -100 reward), it experiences a sudden spike in regret compared to the optimal reward of -13.
- **Exploration variance**: Random exploration can lead to suboptimal actions, especially in dangerous areas near the cliff, causing temporary increases in regret.
- **Policy updates**: As the Q-values are updated, the agent may temporarily switch between different policies, leading to fluctuations in performance.
- **Learning instability**: During the early stages of learning, the agent's policy is still evolving, causing occasional poor decisions that manifest as jumps in regret.

These jumps become less frequent as the agent learns and converges to a stable policy.

1.3 Epsilon 0.9 changes linearly. Why? [2.5-points]

Epsilon 0.9 shows linear regret because:

- **High exploration rate**: With $\epsilon = 0.9$, the agent explores 90% of the time, meaning it takes random actions most of the time.
- **Consistent suboptimal behavior**: Since the agent rarely exploits its learned knowledge, it consistently performs worse than optimal, leading to a steady, linear increase in cumulative regret.
- **Limited learning**: The high exploration rate prevents the agent from effectively learning and exploiting good policies, so regret accumulates at a roughly constant rate.

- **Random walk behavior**: The agent essentially performs a random walk through the environment, with regret increasing linearly as it fails to converge to the optimal policy.

The linear trend indicates that excessive exploration (epsilon 0.9) prevents effective learning and policy improvement.

1.4 Compare the policy for epsilon values 0.1 and 0.9. How do they differ, and why do they look different? [2.5-points]

The policies learned with epsilon 0.1 and 0.9 differ significantly:

Epsilon 0.1 Policy:

- Learns a more deterministic, goal-oriented policy
- Shows clear directional preferences (typically moving right toward the goal)
- Avoids dangerous areas near the cliff due to sufficient exploitation of learned safe actions
- Achieves better final performance (mean reward -17 to -21)

Epsilon 0.9 Policy:

- Remains largely random due to high exploration rate
- Shows no clear directional preferences or learned patterns
- Frequently encounters the cliff due to excessive random exploration
- Achieves poor final performance (mean reward -17, but with high variance)

Why they differ:

- **Exploration vs Exploitation balance**: Epsilon 0.1 allows sufficient exploitation to learn and follow good policies, while epsilon 0.9 prevents effective policy learning.
- **Learning efficiency**: Lower epsilon enables faster convergence to stable policies, while higher epsilon maintains random behavior.
- **Risk management**: Epsilon 0.1 learns to avoid dangerous cliff areas, while epsilon 0.9 continues to explore randomly, including dangerous areas.

1.5 In the epsilon decay section, analyze the optimal policy for the row adjacent to the cliff (the lowest row). Then, compare the different learned policies and their corresponding rewards. [2.5-points]

Optimal Policy Analysis for Cliff-Adjacent Row: The row adjacent to the cliff (lowest row) requires careful navigation. The optimal policy should:

- Move rightward toward the goal when possible
- Avoid moving down (which leads to the cliff and -100 reward)

- Use up/down movements only when necessary to reach the goal

Comparison of Different Epsilon Decay Strategies:

Fast Decay (Decay Rate = 0.005):

- Quickly transitions from exploration to exploitation
- Learns safe policies faster, avoiding cliff encounters
- Achieves good performance (mean reward -17)
- May converge to suboptimal policies due to insufficient exploration

Medium Decay (Decay Rate = 0.002):

- Balanced transition between exploration and exploitation
- Allows sufficient exploration to discover optimal policies
- Achieves good performance (mean reward -17)
- Provides good balance between learning speed and policy quality

Slow Decay (Decay Rate = 0.001):

- Maintains high exploration for longer periods
- May discover better policies but takes longer to converge
- Achieves slightly worse performance (mean reward -19)
- Higher variance due to continued exploration

Key Observations:

- All decay strategies eventually learn to avoid the cliff
- Medium decay provides the best balance between exploration and exploitation
- Fast decay converges quickly but may miss optimal policies
- Slow decay explores more but may not converge effectively within the training period

2 N-step Sarsa and N-step Q-learning

2.1 What is the difference between Q-learning and sarsa? [2.5-points]

The fundamental difference between Q-learning and SARSA lies in their update mechanisms and policy evaluation approaches:

Q-Learning (Off-Policy):

- **Update Rule**: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$
- **Target Policy**: Uses the optimal policy (greedy) for the next state, regardless of the action actually taken

- **Learning Objective**: Learns the optimal Q-function Q^* independent of the policy being followed
- **Behavior**: Can learn optimal policy while following any exploratory policy
- **Convergence**: Guaranteed to converge to optimal policy under certain conditions

SARSA (On-Policy):

- **Update Rule**: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
- **Target Policy**: Uses the same policy (including exploration) that was used to select the next action
- **Learning Objective**: Learns Q-values for the policy actually being followed
- **Behavior**: Learns the consequences of the current policy, including exploration
- **Convergence**: Converges to Q-values of the policy being followed

Key Implications:

- **Risk Management**: SARSA learns safer policies in dangerous environments (like CliffWalking) because it accounts for exploration
- **Optimality**: Q-learning finds optimal policies but may be riskier during learning
- **Stability**: SARSA is more conservative and stable, while Q-learning is more aggressive in seeking optimality

2.2 Compare how different values of n affect each algorithm's performance separately. [2.5-points]

Effect of n on SARSA Performance:

n = 1 (1-step SARSA):

- Mean reward: -100 (agent falls off cliff frequently)
- High variance and instability
- Slow convergence due to single-step updates
- Prone to cliff encounters during learning

n = 2 (2-step SARSA):

- Mean reward: -17 (significant improvement)
- Better stability and faster convergence
- Learns to avoid cliff more effectively
- More efficient learning through 2-step lookahead

n = 5 (5-step SARSA):

- Mean reward: -17 (similar to n=2)
- Good stability but diminishing returns
- Longer lookahead helps with cliff avoidance

- May be slower to converge due to longer update delays

Effect of n on Q-Learning Performance:

n = 1 (1-step Q-Learning):

- Mean reward: -13 (optimal performance)
- Fast convergence to optimal policy
- Efficient learning with single-step updates
- Finds shortest path to goal

n = 2 (2-step Q-Learning):

- Mean reward: -13 (maintains optimal performance)
- Similar convergence speed
- Benefits from multi-step returns
- Stable optimal policy learning

n = 5 (5-step Q-Learning):

- Mean reward: -100 (performance degradation)
- Increased cliff encounters
- Longer lookahead may lead to overestimation
- Less stable learning process

Key Observations:

- SARSA benefits significantly from increasing n ($1 \rightarrow 2$), but diminishing returns beyond n=2
- Q-Learning performs optimally with n=1 and n=2, but degrades with n=5
- Multi-step methods help SARSA learn safer policies by considering longer-term consequences
- Q-Learning's optimal performance is robust to moderate n values but sensitive to very high n

2.3 Is a Higher or Lower n Always Better? Explain the advantages and disadvantages of both low and high n values. [2.5-points]

No, higher n is not always better. The optimal n value depends on the environment characteristics and algorithm:

Advantages of Low n Values (n = 1, 2):

- **Fast Updates**: Immediate feedback allows quick policy adjustments
- **Low Variance**: Shorter update horizons reduce variance in target estimates
- **Computational Efficiency**: Less memory and computation required
- **Stability**: More frequent updates can lead to more stable learning

- **Environment Suitability**: Works well in environments where immediate rewards are informative

Disadvantages of Low n Values:

- **Bootstrap Bias**: Single-step updates may not capture long-term consequences
- **Slow Convergence**: May require more episodes to learn complex policies
- **Exploration Issues**: May not effectively learn from delayed rewards

Advantages of High n Values (n = 5, 10+):

- **Long-term Planning**: Better capture of delayed rewards and consequences
- **Reduced Bootstrap Bias**: More accurate estimates of true returns
- **Better Exploration**: Can learn from sequences of actions
- **Complex Environments**: Effective in environments with sparse or delayed rewards

Disadvantages of High n Values:

- **High Variance**: Longer update horizons increase variance in estimates
- **Computational Cost**: More memory and computation required
- **Delayed Learning**: Updates occur less frequently, slowing initial learning
- **Overestimation Risk**: May lead to overoptimistic value estimates
- **Environment Sensitivity**: Performance can degrade in certain environments

Optimal n Selection Guidelines:

- **Environment Complexity**: Use higher n for complex environments with delayed rewards
- **Algorithm Type**: SARSA benefits more from moderate n values, Q-Learning often works best with n=1
- **Exploration Strategy**: Higher n helps when exploration is risky (like CliffWalking)
- **Computational Resources**: Balance between performance and computational cost
- **Empirical Testing**: Always validate n choice through experimentation

Conclusion: The optimal n value is environment and algorithm-specific. For CliffWalking, n=2 appears optimal for SARSA, while n=1-2 works best for Q-Learning. The key is finding the right balance between bias reduction (higher n) and variance control (lower n).

3 DQN vs. DDQN

3.1 Which algorithm performs better and why? [3-points]

Based on the experimental results, **DDQN performs significantly better than DQN** in the CartPole environment:

Performance Comparison:

- **DDQN**: Mean reward 477.33 ± 1027.56 (Agent 2), with several agents achieving rewards 450
- **DQN**: Mean reward 153.33 ± 16.89 (Agent 0), with no agents achieving the 450+ threshold
- **Success Rate**: DDQN had agents that passed the evaluation threshold, while DQN failed to meet the minimum requirement

Why DDQN Performs Better:

1. **Reduced Overestimation Bias**:
   - DQN suffers from overestimation bias due to the max operator using the same network for both action selection and evaluation
   - DDQN decouples action selection (online network) from action evaluation (target network), reducing this bias
2. **Improved Stability**:
   - DDQN's dual-network architecture provides more stable learning targets
   - Soft target network updates ($= 0.008$) ensure gradual, stable parameter updates
   - Reduces harmful positive feedback loops that plague DQN
3. **Better Convergence**:
   - DDQN converges to more accurate Q-value estimates
   - More reliable policy learning due to reduced variance in target estimates
   - Better handling of the exploration-exploitation trade-off
4. **Enhanced Sample Efficiency**:
   - More accurate Q-values lead to better action selection
   - Reduced need for excessive exploration due to better value estimates
   - Faster learning of optimal policies

3.2 Which algorithm has a tighter upper and lower bound for rewards. [2-points]

DDQN has tighter upper and lower bounds for rewards compared to DQN.

Evidence from Results:

- **DDQN Bounds**: More consistent performance across different random seeds, with tighter confidence intervals
- **DQN Bounds**: Wider variance in performance, indicating less stable learning
- **Visual Analysis**: The smoothed reward plots show DDQN has narrower confidence bands (blue shaded area) compared to DQN (red shaded area)

Why DDQN Has Tighter Bounds:

- **Reduced Variance**: The dual-network architecture reduces variance in Q-value estimates
- **Stable Learning**: Soft target updates prevent sudden policy changes that cause reward fluctuations
- **Consistent Performance**: More reliable convergence leads to consistent performance across different runs
- **Better Generalization**: More accurate Q-values generalize better across different episodes

3.3 Based on your previous answer, can we conclude that this algorithm exhibits greater stability in learning? Explain your reasoning. [2-points]

Yes, DDQN exhibits significantly greater stability in learning compared to DQN.

Evidence Supporting Greater Stability:

1. **Tighter Reward Bounds**:
   - Narrower confidence intervals indicate more consistent performance
   - Less variance in episode rewards across different training runs
   - More predictable learning curves
2. **Reduced Overestimation**:
   - DDQN's decoupled action selection and evaluation prevents overoptimistic Q-values
   - More realistic value estimates lead to more stable policy updates
   - Avoids the positive feedback loops that destabilize DQN learning
3. **Consistent Convergence**:
   - DDQN agents consistently achieve better final performance
   - More reliable convergence to optimal policies
   - Less sensitivity to hyperparameter choices and random seeds
4. **Smoother Learning Curves**:
   - DDQN shows smoother, less erratic reward progression
   - Fewer sudden drops or spikes in performance
   - More gradual and consistent improvement over time

Mechanisms Contributing to Stability:

- **Dual Network Architecture**: Separates action selection from value estimation
- **Soft Target Updates**: Gradual parameter updates prevent sudden policy changes
- **Reduced Bias**: More accurate Q-values lead to more stable learning
- **Better Exploration**: More informed exploration reduces random performance fluctuations

3.4 What are the general issues with DQN? [2-points]

DQN suffers from several fundamental issues that limit its performance and stability:

1. **Overestimation Bias**:
   - The max operator in Q-learning uses the same network for both action selection and evaluation
   - Leads to systematic overestimation of Q-values
   - Causes suboptimal action selection and unstable learning
2. **Instability in Learning**:
   - Single network architecture creates harmful correlations between target and predicted values
   - Positive feedback loops can cause Q-values to diverge
   - High variance in learning updates leads to erratic performance
3. **Correlation Issues**:
   - Consecutive experiences are highly correlated
   - Can lead to catastrophic forgetting of previously learned information
   - Makes learning inefficient and unstable
4. **Non-stationary Targets**:
   - Target values change as the network parameters update
   - Creates moving target problem that hinders convergence
   - Leads to oscillating or diverging behavior
5. **Exploration Challenges**:
   - Fixed -greedy exploration may not be optimal
   - Can get stuck in suboptimal policies
   - Difficulty balancing exploration and exploitation effectively
6. **Sample Inefficiency**:
   - Requires many samples to learn effective policies
   - Slow convergence in complex environments
   - Poor performance in environments with sparse rewards

3.5 How can some of these issues be mitigated? (You may refer to external sources such as research papers and blog posts be sure to cite them properly.) [3-points]

Several techniques have been developed to address DQN's limitations:

1. **Double DQN (DDQN)** [4]:
   - Decouples action selection from action evaluation using separate networks
   - Reduces overestimation bias significantly
   - Improves stability and performance in most environments
2. **Prioritized Experience Replay** [5]:
   - Samples important transitions more frequently
   - Improves sample efficiency and learning speed
   - Reduces correlation issues by focusing on informative experiences
3. **Dueling DQN** [6]:
   - Separates value function and advantage function estimation
   - Better handling of environments where action choice doesn't significantly affect state value
   - Improved learning efficiency and policy quality
4. **Distributional RL (C51, QR-DQN)** [7]:
   - Learns the full distribution of returns instead of just expected values
   - Provides richer information for decision-making
   - Can improve performance in stochastic environments
5. **Noisy Networks** [8]:
   - Replaces -greedy exploration with learned noise in network weights
   - More sophisticated exploration strategy
   - Can improve exploration efficiency
6. **Rainbow DQN** [9]:
   - Combines multiple improvements: DDQN, Prioritized Replay, Dueling, Distributional RL, Noisy Networks, and N-step returns
   - State-of-the-art performance on Atari games
   - Demonstrates the cumulative benefits of multiple improvements
7. **Target Network Updates**:
   - Periodic or soft updates of target network parameters
   - Reduces non-stationarity of targets

- Improves learning stability

3.6 Based on the plotted values in the notebook, can the main purpose of DDQN be observed in the results? [2-points]

Yes, the main purpose of DDQN can be clearly observed in the results.

Evidence of Overestimation Reduction:

1. **Q-Value Comparison**:
   - DDQN shows more conservative, realistic Q-value estimates
   - Lower maximum Q-values compared to DQN's potentially overestimated values
   - More stable Q-value evolution over time
2. **Performance Stability**:
   - DDQN achieves consistent high performance (agents reaching 450+ reward threshold)
   - DQN fails to consistently reach the performance threshold
   - DDQN's success rate demonstrates more reliable learning
3. **Learning Curve Analysis**:
   - DDQN shows smoother, more stable learning progression
   - Less erratic behavior compared to DQN's volatile learning curves
   - More consistent convergence to optimal policies
4. **Variance Reduction**:
   - DDQN exhibits lower variance in performance across different random seeds
   - Tighter confidence intervals in reward plots
   - More predictable and stable learning behavior

Mechanism Verification: The results confirm that DDQN's dual-network architecture successfully:

- Reduces overestimation bias by separating action selection and evaluation
- Provides more stable learning targets
- Enables more reliable convergence to optimal policies
- Improves overall performance consistency

3.7 The DDQN paper states that different environments influence the algorithm in various ways. Explain these characteristics (e.g., complexity, dynamics of the environment) and their impact on DDQN's performance. Then, compare them to the CartPole environment. Does CartPole exhibit these characteristics or not? [4-points]

Environment Characteristics Affecting DDQN Performance:

1. **State Space Complexity**:
   - **High-dimensional states**: Require more sophisticated function approximation
   - **Impact on DDQN**: More complex states increase overestimation bias in DQN, making DDQN's bias reduction more valuable
   - **CartPole**: Low-dimensional (4D continuous state space) - moderate complexity
2. **Action Space Characteristics**:
   - **Discrete vs Continuous**: Discrete actions are easier to handle with Q-learning
   - **Action dimensionality**: More actions increase the max operator's overestimation effect
   - **CartPole**: Binary discrete actions (left/right) - simple action space
3. **Reward Structure**:
   - **Sparse vs Dense rewards**: Sparse rewards make overestimation more problematic
   - **Reward magnitude**: Large reward ranges amplify overestimation effects
   - **CartPole**: Dense rewards (+1 per timestep) with clear success/failure signals
4. **Environment Dynamics**:
   - **Deterministic vs Stochastic**: Stochastic environments increase variance
   - **Episodic vs Continuing**: Episodic tasks have clearer termination conditions
   - **CartPole**: Deterministic dynamics with episodic structure
5. **Exploration Requirements**:
   - **Exploration difficulty**: Environments requiring extensive exploration benefit more from DDQN
   - **Risk of exploration**: Dangerous environments make overestimation more costly
   - **CartPole**: Moderate exploration requirements, low risk

CartPole Environment Analysis:

Characteristics CartPole Exhibits:

- **Moderate State Complexity**: 4D continuous state space provides sufficient complexity for overestimation to occur
- **Clear Success/Failure**: Episodic structure with clear termination conditions

- **Dense Rewards**: Continuous +1 rewards provide immediate feedback
- **Deterministic Dynamics**: Predictable environment behavior
- **Binary Actions**: Simple action space reduces but doesn't eliminate overestimation

Characteristics CartPole Lacks:

- **High-dimensional States**: Unlike Atari games with pixel observations
- **Complex Action Spaces**: Unlike environments with many possible actions
- **Sparse Rewards**: Unlike environments where rewards are rare
- **High Risk Exploration**: Unlike environments where exploration can be catastrophic
- **Stochastic Dynamics**: Unlike environments with significant randomness

Impact on DDQN Performance in CartPole:

- **Moderate Benefit**: CartPole's characteristics provide moderate conditions for DDQN's advantages
- **Overestimation Reduction**: Still beneficial due to continuous state space and function approximation
- **Stability Improvement**: Clear benefits in learning stability and convergence
- **Performance Gains**: Significant but not as dramatic as in more complex environments

Comparison to Other Environments:

- **Atari Games**: DDQN shows much larger improvements due to high-dimensional pixel states and complex action spaces
- **Robotic Control**: DDQN benefits more due to continuous action spaces and sparse rewards
- **Strategic Games**: DDQN shows significant improvements due to complex state spaces and delayed rewards

3.8 How do you think DQN can be further improved? (This question is for your own analysis, but you may refer to external sources such as research papers and blog posts be sure to cite them properly.) [2-points]

Several promising directions for further improving DQN beyond current methods:

1. **Meta-Learning Approaches** [10]:
   - Learn to learn better exploration strategies
   - Adapt hyperparameters during training
   - Transfer knowledge across different environments
2. **Model-Based Extensions**:
   - Combine model-free Q-learning with learned environment models

- Use imagination-based planning for better sample efficiency
- Incorporate uncertainty estimates in model predictions

3. **Multi-Agent Considerations**:
   - Develop algorithms robust to non-stationary environments
   - Handle competitive and cooperative scenarios
   - Learn opponent modeling strategies
4. **Advanced Exploration Strategies**:
   - Implement curiosity-driven exploration [11]
   - Use information-theoretic exploration measures
   - Develop adaptive exploration schedules
5. **Architectural Improvements**:
   - Incorporate attention mechanisms for better state representation
   - Use transformer architectures for sequence modeling
   - Implement memory-augmented networks for long-term dependencies
6. **Regularization Techniques**:
   - Apply dropout and batch normalization more effectively
   - Use spectral normalization for training stability
   - Implement gradient clipping and normalization strategies
7. **Distributional Extensions**:
   - Develop more sophisticated distributional RL methods
   - Incorporate risk-sensitive learning objectives
   - Use quantile regression for better uncertainty quantification
8. **Continual Learning**:
   - Prevent catastrophic forgetting in non-stationary environments
   - Implement elastic weight consolidation techniques
   - Develop lifelong learning capabilities

Promising Research Directions:

- **Neural Architecture Search**: Automatically design optimal network architectures for specific environments
- **Federated Learning**: Train DQN agents across multiple environments simultaneously
- **Quantum Reinforcement Learning**: Explore quantum computing approaches for Q-learning
- **Neurosymbolic Integration**: Combine neural networks with symbolic reasoning

References

[1] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2nd Edition, 2020. Available: http://incompleteideas.net/book/the-book-2nd.html.

[2] Gymnasium Documentation. Available: https://gymnasium.farama.org/

[3] Grokking Deep Reinforcement Learning. Available: https://www.manning.com/books/grokking-deep-reinforcement-learning

[4] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, no. 1, 2016.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv preprint arXiv:1511.05952, 2015.

[6] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in International conference on machine learning, 2016, pp. 1995-2003.

[7] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in International Conference on Machine Learning, 2017, pp. 449-458.

[8] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and A. Legg, "Noisy networks for exploration," arXiv preprint arXiv:1706.10295, 2017.

[9] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[10] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," arXiv preprint arXiv:1611.05763, 2016.

[11] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in International conference on machine learning, 2017, pp. 2778-2787.

[12] Cover image designed by freepik