LLM Agents & Deep Q-Learning with Atari Games (Bank Heist)

Summary

This report presents the implementation and analysis of a Deep Q-Learning (DQN) agent developed for the Atari game *Bank Heist*. Across 2,900 training episodes and five experimental configurations, the project explored how different hyperparameters affect learning efficiency. The study found that adjusting the epsilon decay rate significantly improved learning, producing a 158% performance gain compared to the baseline configuration. The results demonstrate how careful tuning of exploration parameters drives efficiency in complex reinforcement learning tasks.

1. Baseline Implementation and Performance

The baseline configuration provided the foundation for all subsequent experiments. Parameters were adapted from standard DQN literature and optimized for the Bank Heist environment.

Training Parameters

- Total episodes: 1,000
- Maximum steps per episode: 2,000
- Learning rate (alpha): 0.00025
- Discount factor (gamma): 0.99
- Initial epsilon: 1.0
- Minimum epsilon: 0.01
- Epsilon decay rate: 0.995
- Batch size: 32
- Replay memory capacity: 30,000 transitions

Computational Setup

- Platform: Google Colab with Tesla T4 GPU
- Average training time: 90 minutes
- GPU memory usage: approximately 4 GB

Performance Summary

- Episodes 1–400: average reward 27.9 points (early exploration)
- Episodes 401–800: average reward 215.7 points (rapid improvement)
- Episodes 801–1000: average reward 493.0 points (stable convergence)
- Average episode length: 594 steps

- Final epsilon: 0.017

Early in training, the agent behaved randomly, exploring the environment without understanding the game rules. Over time, it learned from experience which sequences of moves led to success, like how a human player improves after many games.

The neural network refined its estimate of the optimal Q-function through repeated gradient updates that minimized the Bellman error. The rapid improvement phase coincided with effective experience replay and reduced exploration, demonstrating convergence toward a stable policy.

2. Environment Analysis

State Space

- Raw observation: $210 \times 160 \times 3$ RGB frames
- Preprocessed frames: grayscale, resized to 84×84, normalized between 0–1
- Temporal stacking: 4 consecutive frames
- Final state shape: (4, 84, 84)
- Total inputs: 28,224 pixels per state

The agent "sees" the last four frames together to understand movement—like watching a short clip rather than a single image.

Action Space

- Discrete actions: 18 total
- Basic: NOOP, FIRE, UP, DOWN, LEFT, RIGHT
- Diagonal: UP-RIGHT, UP-LEFT, DOWN-RIGHT, DOWN-LEFT
- Fire combinations: 8 movement + FIRE variants

Q-Table Feasibility

- Theoretically infinite: impossible to store $|\text{states}| \times |\text{actions}|$
- Replaced with neural approximation: $Q(s, a) \approx \text{NeuralNetwork}\theta(s)[a]$
- Advantages: handles continuous states, generalizes across inputs, requires far less memory
- Trade-offs: introduces approximation error and training instability

3. Reward Structure

Reward Source

- Native game rewards directly from Atari Learning Environment

- Rewards assigned for successful bank robberies
- Sparse and delayed feedback; no intermediate shaping

Why Native Rewards Were Retained

- Authentic representation of actual gameplay objectives
- Avoids bias from hand-crafted rewards
- Demonstrates the DQN's ability to learn from sparse signals
- Simplifies experimental reproducibility

Reward Statistics

- Median reward: 450 points
- Maximum reward: 850 points
- Minimum reward: 0 points
- Standard deviation: 180 points

The agent only receives rewards when robbing banks, not for simply surviving. It must link long action sequences with eventual success, a difficult but realistic learning challenge.

4. Bellman Equation Parameter Testing

Baseline Parameters

- Alpha (learning rate): 0.00025
- Gamma (discount factor): 0.99

Experiment 1 – Higher Learning Rate

- $\alpha = 0.0005$
- Average reward: 8.8 (68% lower than baseline)
- Result: unstable learning, large gradient oscillations

Experiment 2 – Lower Discount Factor

- $\gamma = 0.95$
- Average reward: 23.0 (18% lower than baseline)
- Result: short-sighted decisions, weaker long-term planning

Conclusions

- High learning rates caused divergence by overshooting gradients

- Lower discount factors prevented effective future planning
- Baseline ($\alpha = 0.00025$, $\gamma = 0.99$) offered best stability and long-term performance

When learning too fast, the agent "forgets" old lessons with each new update. When it values the future too little, it becomes impatient and fails to plan routes effectively.

5. Policy Exploration

Baseline: Epsilon-Greedy Policy

- Random exploration probability: ε
- ε decreases from $1.0 \rightarrow 0.01$ with decay = 0.995
- Early episodes emphasize exploration; later episodes exploit learned Q-values

Alternative: Boltzmann (Softmax) Policy

- Probability of choosing action a: $P(a|s) = \exp(Q(s,a)/T) / \Sigma \exp(Q(s,a')/T)$
- Temperature T controls randomness
- T decays gradually to reduce exploration

Results

- Boltzmann exploration: average reward 38.7 (+39% vs early baseline)
- ε-greedy with slower decay (see Section 6): average reward 71.9 (+158%)

Insights

- Boltzmann produced moderate improvement with higher computation cost
- ε-greedy achieved superior results through optimized decay schedule
- Discrete action spaces like Atari benefit more from ε-greedy simplicity

6. Exploration Parameter Optimization

Baseline Values

- Initial ε: 1.0
- Minimum ε: 0.01
- Decay: 0.995

Experiment 3 – Slower Epsilon Decay

- Decay: 0.99
- Average reward: 71.9 (158% higher)

- Explanation: longer exploration period before convergence

Epsilon Progression Comparison

- Episode 100: baseline ε = 0.606, slower decay ε = 0.366
- Episode 400: baseline ε = 0.135, slower decay ε = 0.018
- Episode 1000: both reach near-minimum

Insights

- Extended exploration produced richer experience replay
- Slower decay allowed discovery of better long-term strategies
- Improved stability and sample efficiency

Allowing the agent to explore longer before settling helped it find better paths, similar to how exploring a city thoroughly reveals shortcuts.

7. Performance Metrics

Average Episode Length

- Mean: 594 steps
- Maximum: 838 steps
- Minimum: 364 steps

Early vs Late Performance

- Early episodes: 580 steps, average 27.9 reward
- Late episodes: 594 steps, average 493 reward
- Reward-per-step increased from 0.048 → 0.829

Interpretation

- The agent learned efficient goal-directed behavior rather than simply surviving longer
- Stable episode lengths with rising rewards indicate improved decision efficiency

8. Q-Learning Classification

Q-Learning is a **value-based** algorithm that learns the expected cumulative reward for each state–action pair.

Key Properties

- Learns Q(s,a) using the Bellman update
- Derives policy implicitly: π(s) = argmax_a Q(s,a)

- Off-policy: can learn from past experiences
- Ideal for discrete action spaces

Comparison with Policy-Based Methods

- Policy-based algorithms (e.g., PPO, REINFORCE) optimize π_θ(a|s) directly
- Q-Learning uses regression to update Q-values
- Actor-critic methods combine both approaches

Insights

- DQN used here is value-based
- Simplifies computation and improves sample efficiency
- Deterministic policies appropriate for Atari-type environments

9. Deep Q-Learning vs LLM Agents

Key Differences

- DQN learns by interaction; LLMs learn from massive text data
- DQN requires millions of steps per task; LLMs generalize across tasks
- DQN outputs discrete control actions; LLMs output language tokens
- DQN lacks semantic understanding; LLMs reason abstractly

Parallel Concepts

- Reward signals: DQN from environment, LLM from human feedback (RLHF)
- Exploration: DQN uses ε-greedy; LLMs use temperature and top-p sampling
- Policy optimization: DQN minimizes Bellman error; LLMs maximize expected human preference reward

Integration Potential

- LLMs for reasoning and planning
- DQNs for precise low-level control
- Combined systems could merge strategy with execution for advanced agents

10. Bellman Equation and Expected Lifetime Value

Bellman Optimality Equation
Q(s,a) = E[r + γ max_a′ Q(s′,a′)]

Key Components

- r: immediate reward
- γ: discount factor for future rewards
- max_a′ Q(s′,a′): value of best possible next action

Expected Lifetime Value

- Sum of discounted future rewards over the entire episode
- Represents long-term return, not just immediate gains

Example Interpretation

- Choosing a safer route to multiple banks may yield higher total reward than a risky quick robbery
- The agent evaluates all possible futures, choosing the path with the highest expected cumulative value

11. Reinforcement Learning Concepts for LLM Agents

Shared Foundations

- Both use reward-driven optimization
- Both balance exploration and exploitation
- Both face credit-assignment challenges

Direct Parallels

- DQN's reward signal ↔ RLHF's human preference score
- Experience replay ↔ mixed fine-tuning datasets
- Epsilon decay ↔ temperature annealing during sampling

Lessons from This Work for LLMs

- Gradual exploration decay improves learning (Exp. 3 insight)
- Careful learning-rate tuning avoids instability (Exp. 1 result)
- High discount factors maintain long-term coherence in dialogue tasks

12. Planning in Reinforcement Learning and LLMs

Planning in DQN

- Implicit in Q-values; the network learns long-term consequences without explicit simulation
- Fast and reactive but not explainable

Planning in LLM Agents

- Explicit through reasoning chains ("think-step-by-step")

- Observable and interpretable
- Examples: Chain-of-Thought, ReAct, Tree-of-Thought frameworks

Comparison Summary

- RL: compact, numerical, non-interpretable planning
- LLM: linguistic, explainable, slower but transparent reasoning

13. Q-Learning Algorithm Explanation

The Q-learning algorithm is a model-free reinforcement learning technique that learns the optimal action-value function through iterative updates based on the Bellman equation.

Pseudocode

```
Initialize Q(s,a) arbitrarily for all state-action pairs
For each episode:
  Initialize state s
  For each step in episode:
    Choose action a from state s using policy derived from Q (ε-greedy)
    Take action a, observe reward r and next state s'
    Update Q-value:
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max\_a' Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$
  Until s is terminal
```

Mathematical Foundation

The core update equation is derived from the Bellman optimality equation:

Bellman Optimality Equation:

$Q^*(s,a) = E[r + \gamma \max\_a' Q^*(s',a')]$

Temporal Difference (TD) Error:

$\delta = r + \gamma \max\_a' Q(s',a') - Q(s,a)$

Q-Learning Update Rule:

$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \delta$

Where:

- α (alpha) = learning rate, controls how quickly new information overrides old
- γ (gamma) = discount factor, determines importance of future rewards
- r = immediate reward from taking action a in state s
- max_a' Q(s',a') = maximum Q-value achievable from next state s'

Deep Q-Learning Modification

Traditional Q-learning uses a table to store Q(s,a) for each state-action pair. Deep Q-Learning replaces this table with a neural network:

Q(s,a) ≈ NeuralNetwork_θ(s)[a]

Key Enhancements:

1. Experience Replay: Store transitions (s,a,r,s') in memory buffer; sample random batches to break temporal correlations
2. Target Network: Separate network with frozen parameters for calculating target Q-values; updated periodically to stabilize training
3. Loss Function: Mean squared error between predicted Q-values and target Q-values

Training Process:

```
Sample minibatch of transitions from replay memory
For each transition (s,a,r,s',done):
  if done:
    target = r
  else:
    target = r + γ max_a' Q_target(s',a')
  loss = (Q_policy(s)[a] - target)²
Backpropagate loss and update policy network parameters
Periodically copy policy network weights to target network
```

This approach allows the algorithm to handle high-dimensional state spaces like raw pixel inputs while maintaining training stability through experience replay and target networks.

14. Deep Q-Learning and LLM Integration

Integrating Deep Q-Learning with Large Language Model-based systems presents opportunities for creating more sophisticated agents that combine precise control with semantic understanding.

Potential Architecture Approaches

1. Hierarchical Control Architecture

- **LLM Layer:** High-level strategic planning and goal decomposition
- **DQN Layer:** Low-level action execution and motor control
- **Flow:** LLM interprets objectives → generates subgoals → DQN executes optimal policies

Example Implementation:

User Command: "Collect all treasures on the left side of the map first"

↓

LLM Processing: Parses natural language → identifies spatial constraints

↓

Subgoal Generation: [Navigate to left region, collect items sequentially]

↓

DQN Execution: Uses learned Q-values to move efficiently, avoid obstacles

↓

Feedback Loop: DQN reports progress → LLM adjusts strategy if needed

2. *LLM as Reward Shaper*

- Problem: Sparse rewards make learning difficult
- Solution: LLM provides interpretable auxiliary rewards
- Process: LLM analyzes game state → generates semantic feedback → DQN incorporates into reward signal

Example:

Game State: Agent moves closer to treasure

LLM Analysis: "Progress toward objective detected"

Auxiliary Reward: +0.1 (in addition to game's native reward)

Result: Denser feedback accelerates learning

3. Hybrid Policy System

- Concept: Blend LLM recommendations with DQN learned policies
- Formula: $\pi\_hybrid(s) = \lambda \cdot \pi\_LLM(s) + (1-\lambda) \cdot \pi\_DQN(s)$
- Advantage: LLM handles novel situations; DQN optimizes known scenarios

Gaming Assistants:

- LLM understands player requests in natural language
- DQN executes precise in-game actions
- Example: Voice command "Find health packs" → DQN navigates efficiently

Robotics:

- LLM interprets task descriptions
- DQN controls motor actions
- Example: "Clean the kitchen" → LLM plans sequence → DQN executes movements

Autonomous Systems:

- LLM reasons about environment conditions
- DQN makes real-time control decisions
- Example: Self-driving car receives LLM route plan, DQN handles steering/acceleration

Implementation Considerations

Challenges:

- Latency: LLM inference slower than DQN forward pass
- Consistency: Ensuring LLM and DQN objectives align
- Training: How to jointly optimize both systems

Solutions:

- Use LLM for sparse strategic decisions, DQN for dense tactical control
- Train DQN with LLM-guided exploration
- Implement verification layer to reconcile conflicting outputs

This integration approach combines the strengths of both paradigms: LLM's semantic reasoning and generalization with DQN's precise, learned control policies.

15. Code Attribution

Adapted from External Sources

- DQN Architecture: Network structure and layer dimensions from Mnih et al. (2015) Nature paper
- Preprocessing Pipeline: 84×84 grayscale conversion and 4-frame stacking from DQN paper

- Replay Memory: Circular buffer and random sampling from standard RL tutorials
- Bellman Update: Q-learning formula from Sutton & Barto textbook
- Double DQN: Action selection/evaluation split from Van Hasselt et al. (2016)

Original Work

- Custom training infrastructure with dual-checkpoint system
- Five experimental configurations with systematic hyperparameter testing
- Memory-efficient trainer design to prevent Colab crashes
- Complete performance analysis across 2,900 episodes
- Visualization and comparative analysis
- Recovery/resumption logic for long training sessions
- Custom metric tracking and logging system
- Integration of all components into cohesive training pipeline
- All experimental design, parameter selection, and conclusions

Conclusion

The Deep Q-Learning agent for *Bank Heist* successfully demonstrated the core mechanisms of reinforcement learning value estimation, policy improvement, and exploration control. Experiments confirmed that exploration scheduling is the most influential factor in early performance and that small parameter changes can drastically alter outcomes. The findings not only align with theoretical expectations of DQN but also reveal direct parallels with reinforcement learning techniques used in modern LLM training. This project highlights how foundational reinforcement learning principles scale from pixel-based environments to intelligent agent architectures.