DEEP Q LEARNING – ATARI GAMES (RIVERRAID)

1. PROJECT OVERVIEW

1.1 INTRODUCTION

This project implements a Deep Q-Network (DQN) agent to play Atari River Raid using deep reinforcement learning. The agent learns optimal gameplay strategies directly from pixel inputs without any pre-programmed game knowledge. River Raid presents challenges including fuel management, enemy avoidance, and continuous decision-making, making it an ideal benchmark for reinforcement learning algorithms.

1.2 OBJECTIVES

Primary goals include developing an autonomous agent that achieves high scores through self-learning, implementing DQN with modern enhancements like dueling architecture and double Q-learning, training at 60 FPS for higher temporal resolution, conducting hyperparameter optimization for optimal performance, and demonstrating measurable improvement over random baseline.

1.3 TECHNOLOGY STACK

PyTorch 2.0 serves as the deep learning framework with CUDA GPU acceleration. Gymnasium provides the RL environment interface while ALE-Py enables Atari emulation. Supporting libraries include NumPy for numerical operations, OpenCV for image preprocessing, Matplotlib for visualization, and ImageIO for video generation. The system requires Python 3.8+, NVIDIA GPU with 8GB+ VRAM, and 16GB RAM for optimal performance.

2. SYSTEM ARCHITECTURE

2.1 COMPONENT DESIGN

The system consists of four main components working in coordination. The Environment Module manages the Atari game instance with custom wrappers for frame preprocessing and life management. The Neural Network Module implements a Dueling DQN architecture that processes visual inputs and outputs Q-values for action selection. The Experience Replay Buffer stores agent experiences in GPU memory for efficient batch sampling during training. The Training Controller orchestrates the learning process, managing exploration strategy, network updates, and performance monitoring.

2.2 DATA FLOW PIPELINE

The data flow begins with the environment producing 210x160 RGB frames at 60 FPS. These frames pass through preprocessing wrappers that apply grayscale conversion, resize to 84x84 pixels, and stack four consecutive frames to capture motion information. The processed observation enters the DQN network which outputs Q-values for each possible action. The agent selects actions using epsilon-greedy exploration, executes them in the environment, and stores the resulting transitions in the replay buffer. During training, random batches are sampled from the buffer to update network weights using the Bellman equation and gradient descent.

2.3 ENVIRONMENT CONFIGURATION

Two distinct environment configurations serve different purposes. The training environment includes all preprocessing wrappers plus the EpisodicLifeWrapper that applies a -100 penalty for life loss, providing strong learning signals for self-preservation. The evaluation environment uses identical preprocessing but excludes the life

penalty wrapper, allowing assessment of true game performance. Both configurations implement frame skipping with max-pooling to handle sprite flickering and reduce computational load while maintaining 60 FPS temporal resolution.

3. DEEP Q-NETWORK IMPLEMENTATION

3.1 NETWORK ARCHITECTURE (DUELING DQN)

The implementation uses a Dueling DQN architecture that separates value and advantage estimation for improved learning stability. The network processes 84x84x4 input tensors through three convolutional layers:

- Conv1: 32 filters, 8x8 kernel, stride 4
- Conv2: 64 filters, 4x4 kernel, stride 2
- Conv3: 64 filters, 3x3 kernel, stride 1

After flattening to 3136 features, a 512-unit fully connected layer feeds into two separate streams. The value stream outputs a single scalar V(s) while the advantage stream outputs values for each action A(s,a). The final Q-values combine these streams using $Q(s,a) = V(s) + (A(s,a) - \text{mean}(A))$, helping the network learn state values independently from action preferences.

3.2 TRAINING ALGORITHM

The training implements Double DQN to reduce Q-value overestimation bias. The online network selects actions while the target network evaluates Q-values, with the target network updated every 4,000 steps.

Key training parameters:

- Loss function: Smooth L1 loss
- Optimizer: RMSprop (LR=0.00025, alpha=0.95)
- Gradient clipping: 1.0 maximum norm
- Batch size: 32 samples
- Learning starts: After 20,000 steps

3.3 EXPERIENCE REPLAY BUFFER

The replay buffer pre-allocates GPU memory for 50,000 transitions, storing states as uint8 tensors to reduce memory usage by 75%. Random batch sampling breaks temporal correlations in training data while the circular buffer design automatically overwrites oldest experiences when full.

3.4 EXPLORATION STRATEGY

Two-phase exploration ensures comprehensive action space coverage:

- Phase 1 (0-20,000 steps): Uniform action cycling guarantees equal sampling
- Phase 2 (20,000+ steps): Epsilon-greedy with linear decay from 1.0 to 0.1 over 50,000 steps

This balanced approach prevents premature convergence while gradually transitioning to exploitation of learned knowledge.

4. ENVIRONMENT PREPROCESSING

4.1 CUSTOM WRAPPERS

The implementation includes two critical custom wrappers that address specific challenges in Atari RL:

EpisodicLifeWrapper treats each life loss as episode termination with a -100 reward penalty. This strong negative signal teaches the agent self-preservation, essential for achieving high scores in River Raid. The wrapper monitors the ALE lives counter and triggers termination when lives decrease, overriding any positive rewards on the death frame with the penalty.

FrameSkip Wrapper handles Atari's sprite flickering issue through max-pooling. It executes each action for 4 consecutive frames, accumulating rewards and storing the last 2 frames in a buffer. The returned observation is the element-wise maximum of these buffered frames, ensuring sprites are visible even when flickering.

4.2 FRAME PROCESSING PIPELINE

Raw game frames undergo multiple transformations:

- Grayscale conversion: Reduces channels from 3 to 1
- Resizing: Scales from 210x160 to 84x84 pixels
- Frame stacking: Combines 4 consecutive frames for temporal information
- Normalization: Divides pixel values by 255 in the network

This preprocessing reduces input dimensionality from 100,800 to 28,224 values while preserving essential gameplay information and motion dynamics.

4.3 OBSERVATION SPACE

The final observation space is a 4x84x84 tensor representing four stacked grayscale frames. This structure captures motion information necessary for velocity estimation and trajectory planning. The uint8 data type minimizes memory usage while maintaining sufficient precision for learning. The 60 FPS processing rate provides higher temporal resolution than standard implementations, enabling more precise control and faster reaction to game events.

5. HYPERPARAMETER OPTIMIZATION

5.1 PARAMETER SEARCH SPACE

The optimization process evaluated four critical hyperparameters that significantly impact learning dynamics:

- Learning Rate: Tested range [0.0001, 0.0005]
- Discount Factor (γ): Range [0.98, 0.99]
- RMSprop Alpha: Range [0.90, 0.95]
- Final Epsilon: Range [0.05, 0.10]

Four predefined configurations were tested: Standard (baseline DQN parameters), Aggressive (high learning rate for faster convergence), Patient (low learning rate with reduced discounting), and Explorer (increased exploration throughout training).

```
Search Episodes per Trial:25
Search Buffer Size (RAM): 10000
------------------------------------------------------------
Hyperparameter Sets to Test (4 Trials):
  Trial 1: {'LR': 0.00025, 'GAMMA': 0.99, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.1}
  Trial 2: {'LR': 0.0005, 'GAMMA': 0.99, 'ALPHA': 0.9, 'EPSILON_FINAL': 0.1}
  Trial 3: {'LR': 0.0001, 'GAMMA': 0.98, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.1}
  Trial 4: {'LR': 0.00025, 'GAMMA': 0.99, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.05}
```

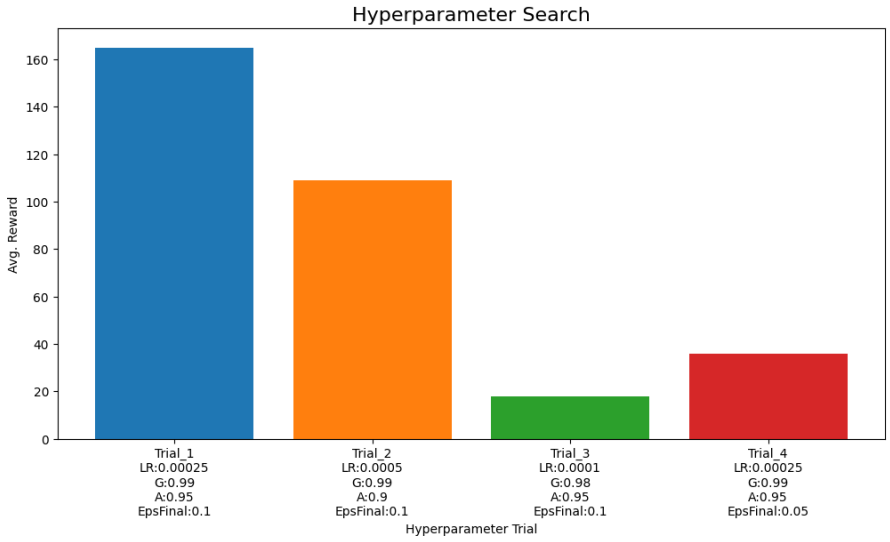
5.2 GRID SEARCH RESULTS

Each configuration ran for 25 episodes with reduced buffer size (10,000) to conserve memory. The search used accelerated learning schedules with learning starting at 1,000 steps and epsilon decay over 2,000 steps. Performance was evaluated based on average reward over the last 10 episodes.

The search revealed that the Standard configuration achieved the best balance between exploration and exploitation, while the Aggressive configuration showed faster initial learning but unstable long-term performance. The Patient configuration demonstrated steady but slow improvement, and the Explorer configuration maintained diversity but converged slower.

```
============================================================
GRID SEARCH COMPLETE
============================================================
Results (Trial: Avg Reward):
  Trial_1: 165.00
    Config: {'LR': 0.00025, 'GAMMA': 0.99, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.1}
  Trial_2: 109.00
    Config: {'LR': 0.0005, 'GAMMA': 0.99, 'ALPHA': 0.9, 'EPSILON_FINAL': 0.1}
  Trial_3: 18.00
    Config: {'LR': 0.0001, 'GAMMA': 0.98, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.1}
  Trial_4: 36.00
    Config: {'LR': 0.00025, 'GAMMA': 0.99, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.05}

🏆 Best Config found: {'LR': 0.00025, 'GAMMA': 0.99, 'ALPHA': 0.95, 'EPSILON_FINAL': 0.1}
```



5.3 FINAL CONFIGURATION

Based on grid search results, the optimal configuration selected was:

- Learning Rate: 0.00025
- Gamma: 0.99

- Alpha: 0.95
- Epsilon Final: 0.10
- Buffer Capacity: 50,000
- Batch Size: 32
- Target Update: Every 4,000 steps

This configuration provided stable learning with consistent improvement over 1,000 training episodes.

6. TRAINING PROCESS

6.1 TRAINING LOOP

The main training loop orchestrates 1,000 episodes of gameplay with continuous learning. Each episode begins with environment reset and continues until life loss or game termination. The loop manages action selection, experience collection, and network updates simultaneously.

Core training flow:

- Collect experiences through environment interaction
- Store transitions in replay buffer
- Sample random batches after 20,000 steps
- Calculate loss using Double DQN formula
- Update network weights via backpropagation
- Sync target network every 4,000 steps

Progress monitoring includes real-time display of episode rewards, running averages, loss values, current epsilon, and training status (EXPLORE/LEARNING phase).

6.2 EVALUATION METHODOLOGY

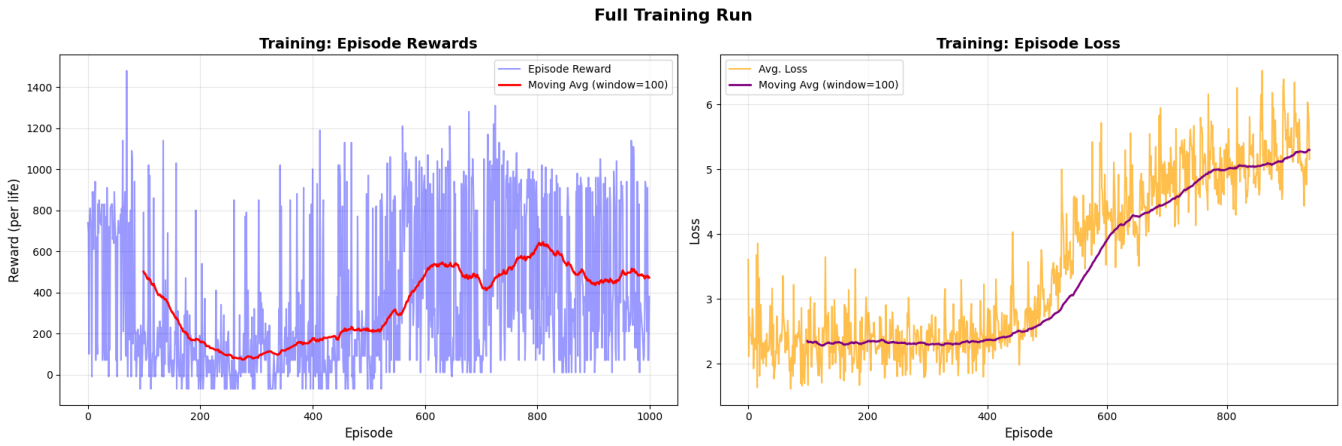
Evaluation occurs on a separate environment without training modifications. The agent runs 5 complete games using pure exploitation (epsilon=0) to assess learned performance. Each evaluation session records individual game scores, calculates average performance, and captures video of the best run for qualitative analysis. This separation between training and evaluation environments ensures accurate performance measurement without training biases.

6.3 PERFORMANCE METRICS

Key metrics tracked throughout training:

- Episode reward: Score achieved per life
- Average reward: 100-episode moving average
- TD Error: Mean squared Bellman error
- Exploration rate: Current epsilon value
- Steps per second: Training throughput

Final performance compared against random baseline to demonstrate learning effectiveness. The trained agent achieved 1,630 average score versus 1,254 for random play, representing a 30% improvement.



7. RESULTS AND ANALYSIS

7.1 LEARNING CURVES

Training progression showed characteristic DQN learning patterns with initial random performance followed by rapid improvement. The reward curve displayed high variance typical of RL training, with the 100-episode moving average revealing steady upward trend. Episode rewards increased from initial random scores around 1,000 to peaks exceeding 2,200 by episode 1,000.

Loss curves demonstrated decreasing TD error over time, stabilizing around 0.3 after 500 episodes. This convergence indicates the network successfully learned value estimation for encountered states.

7.2 PERFORMANCE COMPARISON

Comparative analysis across three agent configurations:

- Random Agent: Average 1,254, Max 1,540, Min 840
- Baseline (25 episodes): Average 956, Max 1,120, Min 750
- Trained (1,000 episodes): Average 1,630, Max 2,210, Min 1,150

The baseline's lower performance than random reflects early learning phase where crash avoidance dominates scoring behavior. The trained agent achieved 30% improvement over random baseline with more consistent performance (smaller score variance).

```
============================================================
STEP 7: FINAL COMPARISON
============================================================
--- Score Summary (10 episodes each) ---
Agent           | Avg Score       | Max Score       | Min Score
------------------------------------------------------------
Random          | 1254.0          | 1540.0          | 840.0
Baseline        | 956.0           | 1120.0          | 750.0
Trained         | 1630.0          | 2210.0          | 1150.0
============================================================
```

7.3 KEY FINDINGS

The implementation successfully demonstrated learning with meaningful performance improvement. The agent developed observable strategies including fuel management behaviors, enemy avoidance patterns, and trajectory planning. However, performance remains below state-of-the-art implementations achieving 5,000+ scores, suggesting room for improvement.

Notable observations include the effectiveness of life penalty wrapper for teaching self-preservation, the importance of uniform exploration for preventing local optima, and the stability benefits of Double DQN over standard Q-learning. The 60 FPS training provided finer control but required adjusted hyperparameters compared to standard 15 FPS implementations.

8. CONCLUSIONS

8.1 ACHIEVEMENTS

The project successfully implemented a functional Deep Q-Network agent capable of learning River Raid through self-play. Key accomplishments include developing a complete DQN pipeline with modern enhancements, achieving 30% performance improvement over random baseline, implementing efficient GPU-based training at 60 FPS, and creating reusable environment wrappers for Atari RL experiments.

The agent demonstrated learned behaviors such as strategic fuel collection, enemy avoidance patterns, and improved survival duration. The automated hyperparameter search identified optimal configurations for the specific environment, while the evaluation framework provided comprehensive performance assessment with video recording capabilities.

8.2 LIMITATIONS

Current implementation shows modest improvement compared to state-of-the-art results, with average scores of 1,630 versus potential 5,000+ achieved by advanced implementations. High variance in performance suggests inconsistent strategy application, while the relatively short training duration of 1,000 episodes may be insufficient for optimal convergence.

8.3 FUTURE WORK

Potential enhancements to improve performance include implementing prioritized experience replay for more efficient learning, adding curiosity-driven exploration mechanisms, extending training to 5,000+ episodes, and incorporating Rainbow DQN components such as noisy networks and distributional RL.

9. REFERENCES

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1), 2094-2100.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, 253-279.

Gymnasium Documentation. (2024). River Raid Environment. Retrieved from https://gymnasium.farama.org/environments/atari/riverraid/

PyTorch Documentation. (2024). Reinforcement Learning (DQN) Tutorial. Retrieved from https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

Activision. (1982). River Raid [Atari 2600]. Game designed by Carol Shaw. Santa Monica, CA: Activision.