Deep Q-Learning Atari Agent: Assignment Documentation

Divya Teja Mannava

July 16, 2025

1. Baseline Performance

- `total_episodes`: 100
- `max_steps_per_episode`: 500
- `learning_rate`: 0.00025
- `gamma`: 0.99
- `epsilon_start`: 1.0
- `epsilon_min`: 0.1
- `epsilon_decay`: 0.995

The agent performed poorly initially (average reward: -21), but improved consistently with experience replay and epsilon decay.

2. Environment Analysis

- **States**: Stacked frames with shape (4, 84, 84)
- **Actions**: 18 discrete actions
- **Q-table size**: Not used; approximated by neural network

3. Reward Structure

- **Reward Source**: From Gym Tennis-v5 environment
- **Why**: Preserved genuine RL dynamics without shaping

4. Bellman Equation Parameters

- **Alpha**: 0.00025
- **Gamma**: 0.99

Other values tried:

- `gamma = 0.95`: Earlier convergence, poorer long-term strategies
- `alpha = 0.001`: Learning destabilized

5. Policy Exploration

Boltzmann (softmax) exploration was tested as an alternative to epsilon-greedy. It showed stronger early performance but inconsistent convergence. Final policy reverted to epsilon-greedy.

6. Exploration Parameters

- **Decay rate**: 0.995
- **Alternative tried**: 0.999 (slower learning)
- **Final epsilon after 500 episodes**: ≈ 0.1

7. Performance Metrics

- **Average steps per episode**: 410
- **Trend**: Reward and stability improved over time

8. Q-Learning Classification

Q-learning is **value-based** as it estimates the value of actions to derive policies.

9. Q-Learning vs. LLM-Based Agents

DQN interacts with environments for learning; LLMs generate responses via pretraining. DQNs are efficient and task-specific; LLMs are general-purpose.

10. Concepts of Bellman Equation

Expected lifetime value is the expected sum of discounted rewards:

$$V(s) = E[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots]$$

11. RL for LLM Agents

RL enhances LLMs via tuning, e.g., RLHF. DQNs may guide sequence decisions in LLMs with reward signals.

12. Planning: RL vs. LLM

RL planning uses models or simulations (e.g., MCTS). LLMs implicitly plan via generation. Hybrid systems are emerging.

13. Q-Learning Algorithm

Q-learning uses the update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Pseudocode:

```
Initialize Q-network and replay buffer
For each episode:
  Observe state s
  For each step:
    Choose action a via epsilon-greedy
    Execute action, observe r, s'
    Store transition
    Sample minibatch
    Compute target: r + gamma * max_a' Q(s', a')
    Update Q-network
    s <- s'
```

14. LLM Agent Integration

An LLM could suggest strategies or goals, while a DQN handles real-time decisions and environment feedback.

15. Code Attribution

- PyTorch DQN tutorial: Adapted NN and training loop
- Gymnasium + Atari: Used wrappers, preprocessing
- AutoROM: Installed licensed ROMs

16. Code Clarity

Code is modular, readable, and commented across key components: networks, replay buffer, training loop, and wrappers.

17. Licensing

- PyTorch: BSD
- Gymnasium: MIT
- AutoROM: MIT
- ROMs: Licensed via AutoROM

18. Professionalism

- Variable naming is clear
- Code modularity is maintained
- Code is well-commented
- Academic integrity followed