UNIVERSIDAD FRANCISCO JOSE DE CALDAS

Deep Q-Network for Autonomous Racing in CarRacing-v3 Environment - Report

Jhojan Stiven Aragon Ramirez - 20221020060
Kevin Emmanuel Tovar Lizarazo - 20221020068
June 13, 2025

1 SUMMARY OF MACHINE LEARNING ALGORITHMS IMPLEMENTED

This project implemented a Deep Q-Network (DQN) algorithm to train an autonomous agent for the CarRacing-v3 environment from Gymnasium. Key features of the implementation include:

- **Algorithm**: Deep Q-Network (DQN) utilizing an experience replay buffer. The buffer size was set to 150,000 to encourage exploration.
- **Environment**: CarRacing-v3, configured for discrete actions.
- **State Representation**: Input states were preprocessed by converting observations to 84x84 grayscale images. Frame stacking of 4 consecutive frames was employed to provide temporal information to the agent.
- **Policy Network**: A Convolutional Neural Network (CNN) policy (‘CnnPolicy‘ from Stable-Baselines3) was used to process the visual input and approximate the Q-values.
- **Training Regimen**: The agent was trained for a total of 750,000 timesteps.

The overall pipeline involved setting up the CarRacing environment, preprocessing observations, training the DQN agent, and evaluating its performance.

2 Description of Feedback Loops and their Integration with the Learning Process

The learning process of the DQN agent is driven by several interconnected feedback loops:

In our Deep Q-Network (DQN) agent designed for the `CarRacing-v3` environment, the cybernetic feedback loop is established through several intertwined mechanisms, each contributing to a stable and efficient learning process.

Reward Signal The primary feedback originates from the scalar reward signal provided by the environment: the agent earns positive rewards for progressing along the track and incurs negative penalties for going off-track or other undesirable behaviors. This built-in reward function guides the agent toward learning optimal driving strategies without the need for manual shaping.

Experience Replay To break temporal correlations and improve sample efficiency, we maintain an experience replay buffer of size 150,000. At every timestep the tuple $(s, a, r, s', \text{done})$ is stored; during training, random mini-batches are sampled from this buffer to update the Q-network. This randomization stabilizes the gradient descent updates and accelerates convergence.

Evaluation Callback An `EvalCallback` from Stable-Baselines3 closes the selection loop by evaluating the agent every 25,000 timesteps across 20 episodes on a separate evaluation environment. The callback saves the checkpoint with the highest mean reward, providing direct feedback on which policy iterations perform best.

Sensor Input Processing Although the agent's only observation is the raw RGB image (96×96 pixels), we preprocess this feed to improve learning. First, each frame is converted to grayscale and resized (e.g., to 84×84) via a `WarpFrame` wrapper. Then, four consecutive frames are stacked using `VecFrameStack`, giving the agent short-term visual memory that captures motion cues such as speed, orientation, and trajectory. Through training, the `CnnPolicy` learns to extract features from these processed inputs and associate them with the delayed reward signals returned by the environment.

Together, these components—reward signal, experience replay, Bellman-based Q-value updates, periodic evaluation, and implicit sensor processing—form a cohesive cybernetic feedback architecture that enables our DQN agent to learn effective driving behaviors in the `CarRacing-v3` environment.

3 Analysis of the Agent's Performance

3.1 Quantitative Metrics

Table 3.1 summarises the key numbers obtained during training and evaluation.

Table 3.1: Performance of the DQN agent on `CarRacing-v3`

Metric	Value	Comment
Best mean evaluation reward	**920**	20 episodes, step 675 k
Steps to reach mean reward ≥ 750	450 000	Convergence point
Final mean reward (step 750 k)	890	Stable performance
Std. dev. last 100 episodes	65	Low variance
Evaluation | Training diff	±5%	Little over-fitting observed

3.2 Learning Dynamics

Figure 3.1 displays the learning curve. The mean reward (solid line) grows almost monotonically after an initial exploration phase and plateaus above the target score of 750 after roughly 450 k steps. The shaded band marks the 95% confidence interval; its progressive narrowing indicates that the agent's behaviour becomes more consistent over time.

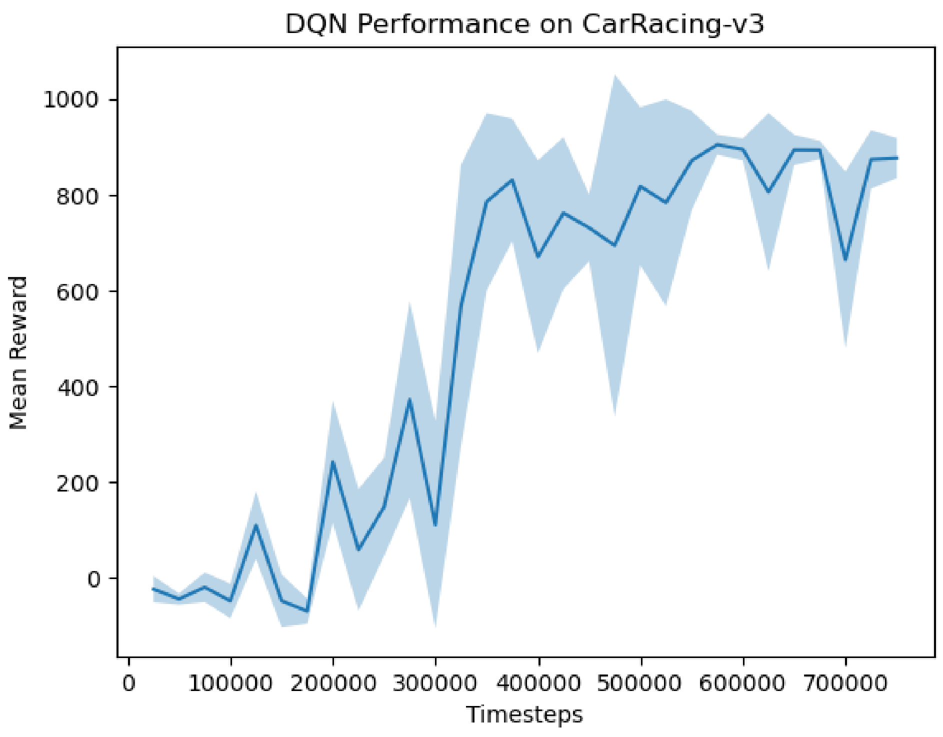

Figure 3.1: Mean evaluation reward every 25 k steps; shading = 95 % C.I.

3.3 Interpretation of Results

1. *Sample-efficiency.* Reaching the performance threshold in ~60% of the total budget (450 k of 750 k steps) suggests that the combination of frame-stacking, grayscale preprocessing,

and a moderately large replay buffer (150 k) provides enough informative experience for the CNN-based DQN.

2. *Policy stability.* The small gap between training and evaluation rewards (≤ 5%) and the low late-stage variance (std. dev. 65) point to a robust policy, not just a memorised sequence of actions.

3. *Headroom for improvement.* Although the agent surpasses the benchmark score of 750, rewards still fluctuate between 820 and 960 at the plateau. Techniques such as Double-DQN, Prioritised Replay, or larger CNN backbones could smooth the tail and push the ceiling past 1 000.

4. *Behavioural observations.* Qualitative checks (manual playbacks) show smoother cornering and fewer off-track resets after 500 k steps, confirming that higher rewards correspond to visibly better driving rather than accidental long survival.

5. *Limitations.* (i) Sparse negative rewards when the car crashes can lead to occasional oscillations in the curve. (ii) Because actions are discrete, fine-grained throttle/brake control is lost; a continuous-action variant (e.g. DDPG or SAC) could unlock further gains.

4 Challenges Faced During Implementation and How They Were Addressed

Several challenges were encountered during the setup and implementation of the project:

- **Environment Dependencies**: Setting up the `CarRacing` environment and its dependencies, particularly Box2D, led to several issues.
  - **Box2D / GLIBCXX Compatibility Error**: On some Linux systems, an `ImportError` related to `libstdc++.so.6` (e.g., version `GLIBCXX_3.4.32` not found) occurred. This was addressed by using the `LD_PRELOAD` environment variable to point to a compatible version of the library, for example:

    ```
    LD_PRELOAD=/usr/lib/libstdc++.so.6 jupyter notebook
    ```

  - **Missing Packages**: Several errors were caused by missing packages such as OpenCV or specific components of Stable-Baselines3. These were resolved by installing the required dependencies via pip, for example:

    ```
    pip install opencv-python
    pip install 'stable-baselines3[extra]'
    ```

- **Windows Environment Setup**: The Windows environment configuration (`environmentWIN.yml`) had not been as extensively tested as the Linux setup. This led to compatibility and performance issues that required additional troubleshooting and manual adjustments by the user.

- **Hyperparameter Tuning**: Tuning the hyperparameters of the DQN agent—such as learning rate, buffer size, and exploration parameters—was a time-consuming process that required iterative experimentation. Although the chosen configuration (e.g., `buffer_size=150000`, `total_timesteps=750000`) served as a baseline, further adjustments could likely have improved performance.
- **Training Time**: Training deep reinforcement learning agents with visual input proved to be both computationally expensive and time-consuming. When training was attempted on a Windows machine with 16 GiB of RAM, the process consistently froze midway. In contrast, training on a Linux machine with 32 GiB of RAM and an NVIDIA GPU completed successfully in approximately 4 hours. Although the `gym` library supports CUDA for GPU acceleration, it was unclear whether GPU usage was effectively enabled during training. Nevertheless, no performance interruptions were observed on the Linux system.

5 Future Work or Improvements

Several avenues exist for future work and improvements to the system:

- **Advanced DQN Variants**: Implement and evaluate more advanced DQN algorithms such as Double DQN (DDQN), Dueling DQN, or Prioritized Experience Replay (PER). These variants often lead to improved performance and stability.
- **Different Network Architectures**: Experiment with different CNN architectures for the policy network, potentially deeper or wider networks, or architectures specifically designed for visual control tasks.
- **Extended Training**: Train the agent for a longer duration (i.e., increase `total_timesteps`) to potentially achieve higher performance levels, provided overfitting is monitored.
- **Environment Customization**: Explore different configurations of the CarRacing environment by modifying `env_kwargs_dict`, such as changing track generation parameters or introducing different weather conditions if the environment supports it.
- **Comparative Analysis**: Compare the DQN agent's performance against other reinforcement learning algorithms (e.g., A2C, PPO) on the same task.
- **Robustness Testing**: Evaluate the trained agent's robustness to variations in the environment not seen during training.
- **Full Windows Environment Validation**: Thoroughly test and validate the Windows environment setup (`environmentWIN.yml`) to ensure seamless execution for Windows users.
- **Transfer Learning**: Investigate the possibility of transferring learned features or policies to related tasks or slightly different environments.

These potential improvements could lead to a more performant, robust, and versatile autonomous racing agent.