# Reinforcement Learning: Training Deep Q-Networks for Autonomous Driving Car

Chandan Jyoti Das
*ECE dept.*
*IITG*
d.chandan@iitg.ac.in

Mir Maiti
*ECE dept.*
*IITG*
mir.maiti@iitg.ac.in

Ishaan Bahl
*EEE dept.*
*IITG*
b.ishaan@iitg.ac.in

***Abstract*—This paper presents the development of a reinforcement learning agent for the CarRacing-v3 game environment using Deep Q-Networks (DQNs). The agent is trained to maximize cumulative rewards by interacting with the environment and learning optimal driving strategies. A convolutional neural network processes image observations while an experience replay buffer stabilizes learning. The training framework includes preprocessing steps, model training, evaluation, and visualization of results. The DQN agent is able to generalize across different tracks and demonstrates stable and intelligent driving behavior, validating the effectiveness of the approach.**

***Index Terms*—Reinforcement Learning, Deep Q-Networks, CarRacing, OpenAI Gym, Artificial Intelligence**

## I. Introduction

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to take actions in an environment to maximize cumulative rewards. Deep Q-Networks (DQNs) combine Q-learning with deep neural networks to learn value functions in high-dimensional state spaces.

In recent years, deep reinforcement learning has been extensively applied to game environments, including Atari games, robotic control, and autonomous navigation. The CarRacing-v3 environment from OpenAI Gym is a popular benchmark due to its visual complexity and continuous state space. This work focuses on developing and training a DQN agent to master this environment by observing and interacting with its surroundings.

## II. Methodology

### A. Environment Setup

CarRacing-v3 simulates a top-down racing environment with randomly generated tracks. The agent receives an RGB image of size 96x96 as observation and chooses from a discrete set of driving actions such as left, right, accelerate, and brake.

The episode ends if the agent leaves the track or completes the course. Rewards are given for forward motion and penalized for off-road driving. The environment is stochastic in its track generation, demanding generalization by the agent across multiple track layouts.

### B. Preprocessing

To reduce computational load and focus on relevant features, the RGB image is converted to grayscale using pixel averaging. The image is then normalized to range [0,1] and reshaped into a channel-first format (1x96x96) compatible with convolutional neural networks. This step is crucial for reducing training time while retaining key spatial features.

### C. Deep Q-Network Architecture

The DQN model architecture includes three convolutional layers for feature extraction followed by two fully connected layers for action value prediction:

- Conv1: 32 filters, 8x8 kernel, stride 4, ReLU
- Conv2: 64 filters, 4x4 kernel, stride 2, ReLU
- Conv3: 64 filters, 3x3 kernel, stride 1, ReLU
- FC1: 512 neurons with ReLU activation
- FC2: Linear layer outputting Q-values for each discrete action

This structure is optimized to process visual input efficiently and make decisions in near real-time.

### D. Experience Replay and Epsilon-Greedy Strategy

An experience replay buffer of size 50,000 stores transition tuples $(s, a, r, s', done)$. Random mini-batches from this buffer are used to update the network, helping stabilize training by reducing the correlation between consecutive experiences.

An epsilon-greedy strategy ensures a balance between exploration and exploitation. Initially, the agent explores with high randomness, but over time, it increasingly exploits learned behavior by choosing the best-known action.

### E. Training Loop and Target Network

The training loop involves resetting the environment, interacting with it for a fixed number of timesteps, storing experiences, and periodically updating the network weights. The key steps include:

- Observation preprocessing and action selection
- Reward and next state observation
- Appending transitions to the replay buffer
- Sampling and training on random batches from the buffer
- Loss calculation using smooth L1 (Huber) loss
- Gradient clipping to improve stability

A target network, updated every few episodes, improves training stability by fixing the targets used in Q-value updates.

### F. Hyperparameter Settings

- Learning Rate: 0.0001
- Discount Factor ($\gamma$): 0.99
- Replay Buffer Capacity: 50000
- Batch Size: 32
- Epsilon Decay: 0.995
- Minimum Epsilon: 0.01
- Target Network Update Frequency: every 2 episodes

### G. Evaluation Strategy

After training, the policy is evaluated using a greedy strategy. The environment is rendered, and frames are captured to generate an MP4 video of the agent's performance. Metrics such as total reward, time steps per episode, and action distribution are logged. These help in understanding the policy behavior and generalization capability.

## III. Experiments and Results

The agent was trained over 200 episodes. The performance was monitored using average episode reward and loss values. Over time, the agent's behavior improved from erratic driving to smooth turns and long trajectories on the road. The learning curve showed a steady rise in average reward.

Model checkpoints were saved periodically. The final evaluation revealed that the agent could complete complex track layouts with high reward accumulation and minimal penalties. Visual inspection of rendered episodes confirmed that the car stayed on track and managed sharp turns effectively.

## IV. Discussion

The results confirm that the DQN approach is effective for the CarRacing-v3 task. The agent was able to adapt to changing tracks, which suggests that the convolutional layers captured robust visual patterns. However, training was sensitive to parameter tuning. Minor changes in learning rate or batch size significantly affected performance.

One notable challenge was ensuring diversity in the experience replay buffer. This was mitigated by using longer episodes and epsilon decay. Future improvements could include using Double DQN to avoid overestimation bias or applying Dueling DQN for better state value estimation.

## V. Contribution

- **Mir Maiti**: Led the implementation of the Q-Learning training pipeline, including the design and coding of the agent's learning algorithm.
- **Ishaan Bahl**: Developed visualization tools to monitor training progress and managed saving and loading of trained models to ensure reproducibility.
- **Chandan Jyoti Das**: Authored the project report and designed the presentation, ensuring clear communication of the project's objectives, methodology, and results.

## VI. Conclusion and Future Work

This paper demonstrated the application of a DQN agent to the CarRacing-v3 task. The agent learned to navigate unseen tracks and make real-time decisions based on grayscale image input. By incorporating key components like experience replay and target networks, the training was stabilized and effective.

Future directions include:

- Exploring more advanced algorithms like PPO, SAC, or A3C
- Introducing visual attention mechanisms or LSTM layers for temporal awareness
- Domain randomization and transfer learning to improve generalization
- Comparison with continuous action methods using the same environment

## Acknowledgment

We would like to thank Prof. Teena Sharma for her invaluable guidance and support throughout this research. We are also grateful to our peers at IIT Guwahati for their cooperation and encouragement.

## References

[1] M. Bojarski et al., "End to End Learning for Self-Driving Cars," arXiv preprint arXiv:1604.07316, 2016.

[2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015.

[3] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction," MIT Press, 2018.

[4] A. Raffin et al., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," Journal of Machine Learning Research, 2021.