Reinforcement Learning Project - Task 1

Idriss Mortadi

idriss.mortadi@student-cs.fr

GitHub Repository

Other Members:
Abdelaziz Guelfane, Abdellah Oumida & Aymane Lotfi

April 2025

Abstract

This project implements and trains a Deep Q-Network (DQN) agent in a predefined simulated driving environment using the Highway-env framework. Key techniques include a replay buffer, epsilon-greedy exploration, and a target network for stable Q-value updates. Training involves optimizing the policy network using the Adam optimizer. Progress is logged using TensorBoard, and results show increasing rewards and episode lengths, demonstrating effective learning dynamics.

Contents

1 Task 1: DQN in a Predefined Environment

1.1 Environment Configuration

The reinforcement learning task uses the `highway-fast-v0` environment. The actions in this environment are **discrete meta-actions** for vehicle control. The environment simulates a 4-lane highway with 15 vehicles and runs for a maximum duration of 60 seconds.

Observation Space

- Type: `OccupancyGrid`
- Features: presence, position (x, y), velocity (vx, vy), heading (cos_h, sin_h)
- Feature ranges: x, y $\in [-100, 100]$, vx, vy $\in [-20, 20]$
- Grid dimensions: $[-20, 20] \times [-20, 20]$ with step size 5×5
- Relative positioning (absolute=False)

Reward Structure

- Collision: -1
- Right lane: $+0.5$ (linearly decreasing for left lanes)
- High speed: $+0.1$ (for speeds in $[20, 30]$ m/s range)
- Lane change: 0

1.2 Deep Q-Network (DQN) Algorithm

The DQN algorithm combines Q-learning with deep neural networks to approximate the action-value function $Q(s, a; \theta)$, aiming to maximize cumulative reward R_t. The update rule follows the Bellman equation:

$$Q(s, a; \theta) = r + \gamma \max_{a'} Q(s', a'; \theta^-). \tag{1}$$

where:

- s and s' are the current and next states, respectively,
- a and a' are the current and next actions, respectively,
- r is the reward received after taking action a in state s,
- $\gamma \in [0, 1]$ is the discount factor,
- θ are the parameters of the policy network,
- θ^- are the parameters of the target network, which are periodically updated to stabilize training.

The loss function used to train the policy network is defined as:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(y - Q(s, a; \theta) \right)^2 \right], \tag{2}$$

where:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-), \tag{3}$$

and $\mathcal{D}$ is the replay buffer containing past experiences.

1.3 Training Procedure

The training procedure for the DQN agent involves the following steps:

1. **Environment Setup:** The agent interacts with a simulated environment, which is configured using the parameters defined in the configuration file.
2. **Network Initialization:** Two neural networks are initialized:
   - The *policy network*, which is used to predict the Q-values for each action.
   - The *target network*, which provides stable targets for the Q-value updates.

The target network is initialized with the same weights as the policy network and is updated periodically.

3. **Replay Buffer:** A replay buffer is used to store experiences (s, a, r, s') collected during interactions with the environment. This buffer enables sampling of mini-batches for training, breaking the temporal correlation between consecutive experiences.

4. **Epsilon-Greedy Policy:** The agent selects actions using an epsilon-greedy policy:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon, \\ \arg\max_a Q(s, a; \theta) & \text{with probability } 1 - \epsilon. \end{cases} \tag{4}$$

   The exploration rate ϵ decays over time to encourage exploitation of the learned policy.

5. **Optimization:** At each step, a mini-batch of experiences is sampled from the replay buffer, and the policy network is updated by minimizing the loss $\mathcal{L}(\theta)$ using the Adam optimizer.

6. **Target Network Update:** The weights of the target network θ^- are updated to match the policy network θ every N steps.

7. **Logging and Evaluation:** Training progress, including rewards, losses, and exploration rate, is logged using TensorBoard. Periodically, the agent's performance is evaluated by recording episodes in a separate evaluation environment.

8. **End of training**: The final policy network is saved, and the agent's performance is recorded by running evaluation episodes in the environment.

Hyperparameters

The training process is governed by several hyperparameters:

- Learning rate (α): Gradient update step size.
- Discount factor (γ): Balances immediate and future rewards.
- Replay buffer size: Number of stored experiences.
- Batch size: Sampled experiences per training step.
- Target network update frequency: Steps between target network updates.
- Exploration parameters (ϵ-start, ϵ-end, ϵ-decay): Exploration-exploitation tradeoff.

Hyperparameters Values

- **Training Control:**
  - `NUM_EPISODES`: `1,000`
  - `BATCH_SIZE`: `64`
  - `TARGET_UPDATE` (steps): `1,000`
- **Q-Learning:**
  - `GAMMA`: `0.99`
  - `LR`: `1e-4`
- **Epsilon-Greedy:**
  - `EPS_START`: `1.0`
  - `EPS_END`: `0.05`
  - `EPS_DECAY`: `0.995`
- **Replay Buffer:**
  - `MEMORY_SIZE`: `5,000,000`

1.4 Techniques Enabling Learning in the Code

The learning process in the provided code is based on the Deep Q-Network (DQN) algorithm, which combines reinforcement learning principles with deep neural networks to solve complex decision-making problems. The following key techniques are employed to enable effective learning:

- **Experience Replay:** The algorithm stores past experiences in a replay buffer and samples mini-batches of experiences during training. This technique breaks the correlation between consecutive experiences and improves the stability of the learning process.
- **Epsilon-Greedy Exploration:** To balance exploration and exploitation, the agent follows an epsilon-greedy policy, where it selects random actions with a probability of ϵ and chooses the action with the highest Q-value otherwise.
- **Optimization Techniques:** Gradient descent-based optimization algorithms, such as Adam, are used to minimize the loss function and update the neural network weights efficiently.
- **Gradient Clipping:** Prevents exploding gradients by clipping during backpropagation, ensuring stable network updates.

1.5 Logging of Training Runs

The code implements comprehensive logging of training runs using TensorBoard and file-based logging. Key mechanisms include:

- **TensorBoard:** Tool to record and save metrics and hyperparameters for each run.
- **Video Recording:** Records the agent's performance post-training using `record_agent.py` and `play_or_record.py` scripts for visual inspection.

1.6 Results and Analysis

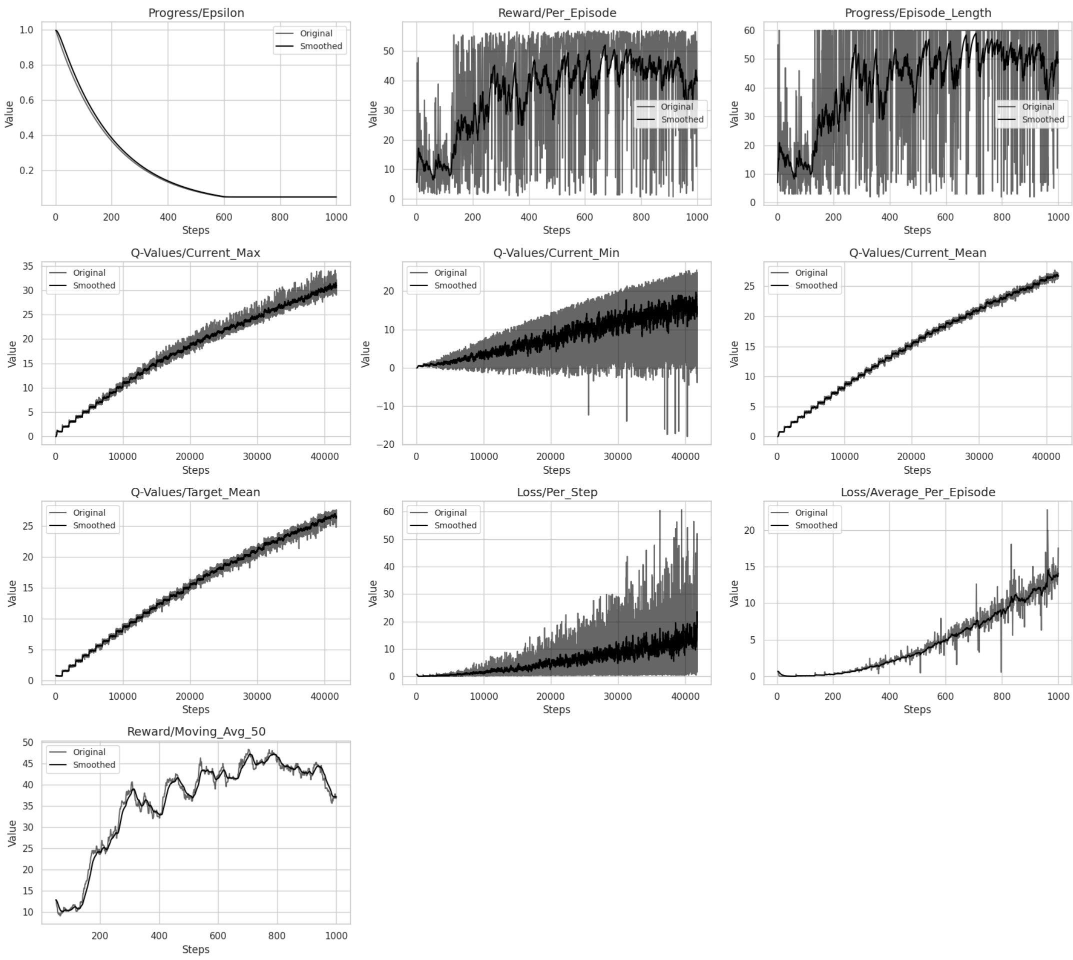

Figure 1: Training Metrics for DQN Agent

The training progress of our Deep Q-Network (DQN) agent is illustrated in Figure 1, showcasing multiple metrics to evaluate performance and learning dynamics over time.

Exploration and Learning Progress

The reward per episode (top-center) shows significant improvement, with the smoothed average increasing from initial values around 10-15 to stabilizing between 35-45. **Episode lengths** (top-right) also increase, indicating the agent's improved ability to survive longer in the environment, correlating with higher rewards.

Q-Value Evolution

The middle and bottom rows display Q-value dynamics. The maximum, minimum, and mean Q-values (middle row) show increasing trends, indicating the agent's optimistic estimations of future rewards. The target mean Q-values (bottom-left) mirror the current Q-values, confirming the target network's successful tracking of the behavior network.

Loss Characteristics

The loss metrics (bottom-center and bottom-right) show an increasing pattern over time, consistent with DQN's learning dynamics. This reflects the agent's continuous adjustment to higher-reward strategies and growing Q-values, not poor learning.

Reward Stability

The moving average reward over 50 episodes (bottom plot) shows a steady increase from approximately 15 to around 37, indicating the agent's learning progress and stabilization towards a consistent policy.

Agent Behavior

A sample video demonstrating the agent's behavior is available here. The agent starts fast and then gradually slows down, tends to stay in the right-most lane, and avoids overtaking. This behavior is due to the reward configuration: a high-speed reward of up to $+0.1$ for speeds between 20–30 m/s, a right-lane reward of $+0.5$, zero reward for lane changes, and a collision penalty of -1.

1.7 Conclusion

These results demonstrate successful training of our DQN agent, with key indicators—increasing rewards, growing episode lengths, and stabilizing Q-values—pointing to effective learning and improved performance.