Applying DQN with Experience Replay to "Dino Chrome" Game

Simone Cappiello

Data Science and Scientific Computing

November 30, 2023

Description of the Game

The Dino game is a simple **endless runner**. The player controls a dinosaur, tasked with **avoiding obstacles** like various-sized cacti and flying birds. The game offers three actions: jump, stay still, and move down. It ends upon touching an obstacle, **the longer the dinosaur survives, the higher the score**.

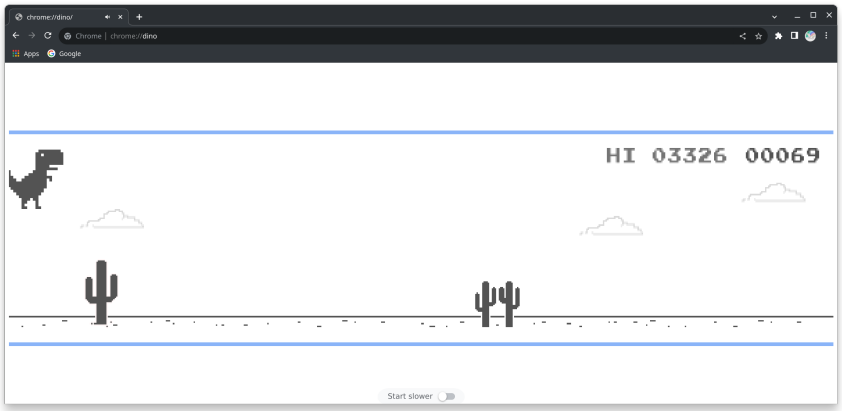


Figure 1: Avoiding a cactus...

Figure 2: ...too early: game over.

Environment Definition

- **Actions**:
  - 3 (discrete) actions:
    1. Jump.
    2. Duck/move down.
    3. No operation, i.e., keep running.
- **States**:
  - Stack of 4 grayscale consecutive frames (to capture motion).
  - Each frame is resized to 84x84 pixels.
- **Rewards Structure**:
  - Reward of $+1$ (variant: $+0.05$) for every frame the dinosaur survives.
  - Penalty of -1 if the dinosaur touches an obstacle (game over).

DQN with Experience Replay

Loss:

$$L_i(\theta_i) = \mathbb{E}_{(s,a)\sim\rho(\cdot)}\left[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2\right]$$

Gradient of Loss:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a)\sim\rho(\cdot),s'\sim\varepsilon}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

Weights Update:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta_i} L_i(\theta_i)$$

Experience Replay:

- Experiences (s_t, a_t, r_t, s_{t+1}) stored in a replay memory.
- Randomly sampled mini-batches from this memory for learning.

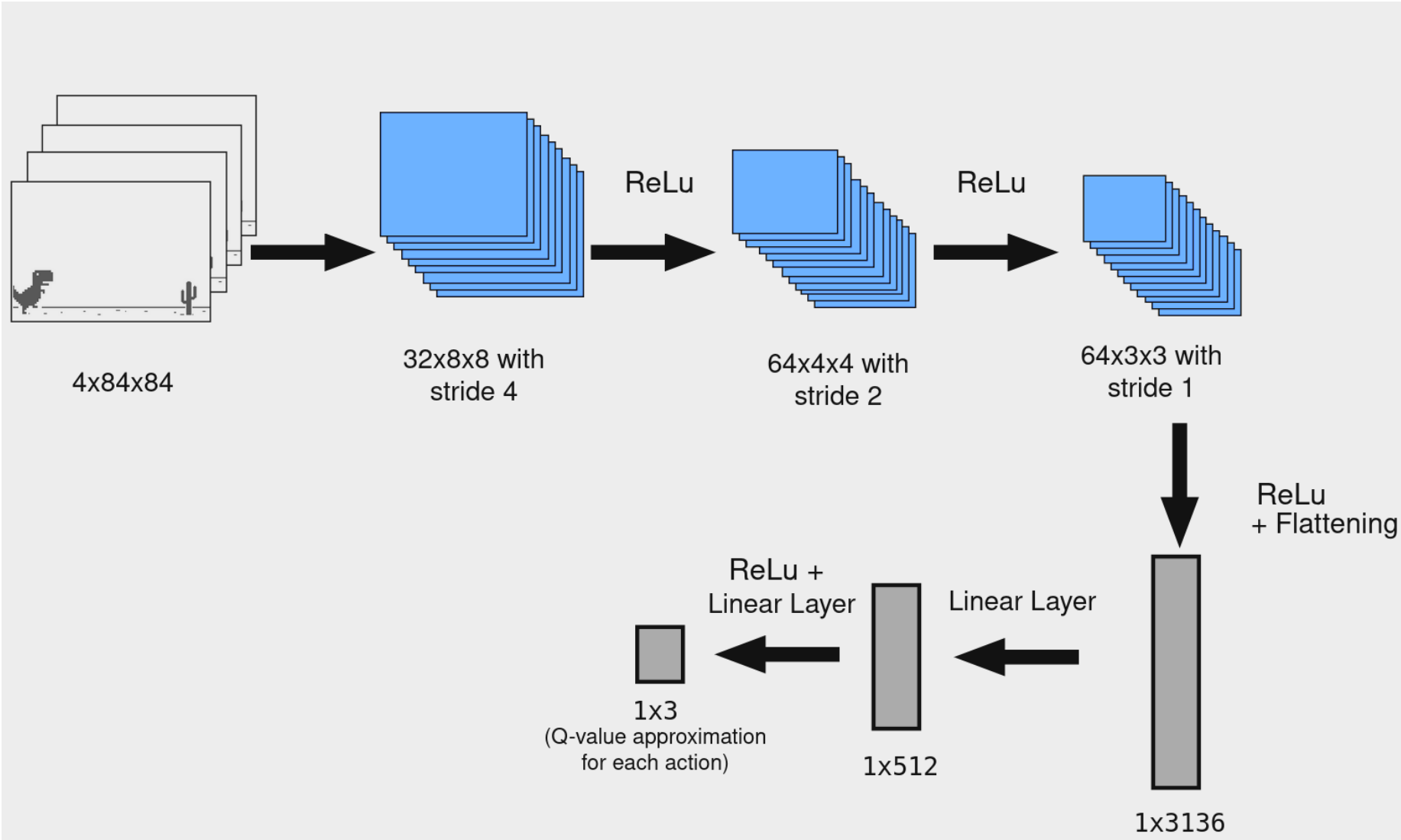
Advantages:

- Enhanced data efficiency.
- Reduced variance.
- Enhanced stability.
- Avoiding feedback loops and parameter divergence.

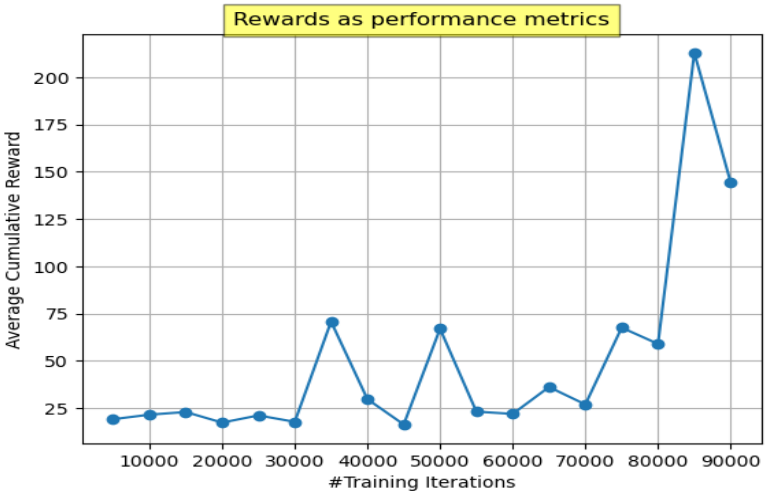
Deep Q-learning with Experience Replay (Algorithm)

- Initialize replay memory D to capacity N
- Initialize action-value function Q with random weights
- **for** episode $= 1$ to M **do**
  - Initialise sequence $s_1 = \{x_1\}$ and $\phi_1 = \phi(s_1)$
  - **for** $t = 1$ to T **do**
    - With probability ϵ select a random action a_t, otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    - Execute a_t in emulator, observe r_t and image x_{t+1}
    - Set $s_{t+1} = s_t, a_t, x_{t+1}$, and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    - Store $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    - Sample minibatch $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    - Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    - Gradient descent on $(y_j - Q(\phi_j, a_j; \theta))^2$

CNN structure



Model 1

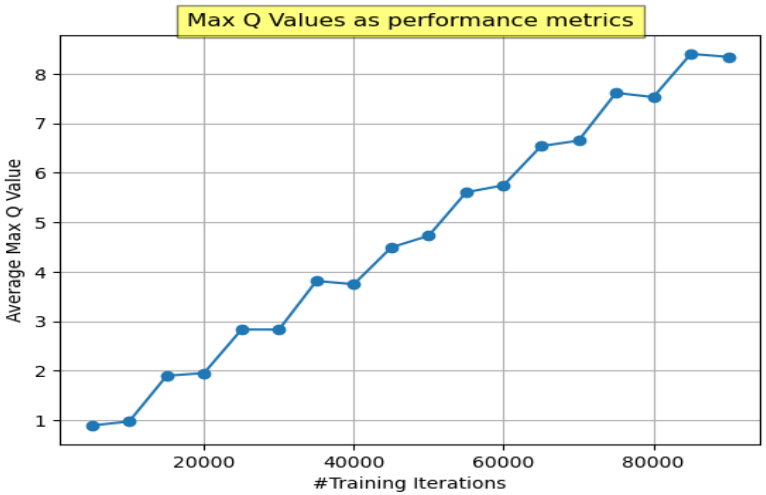


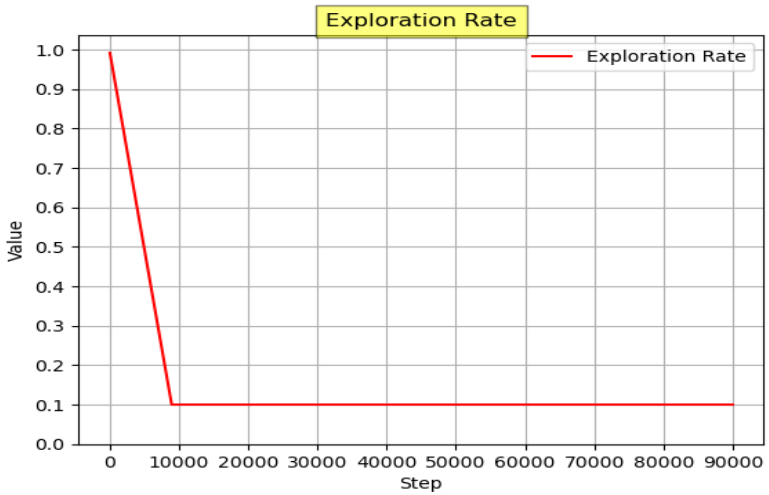
Rewards structure:

- $+1$ for surviving.
- -1 for "game over".

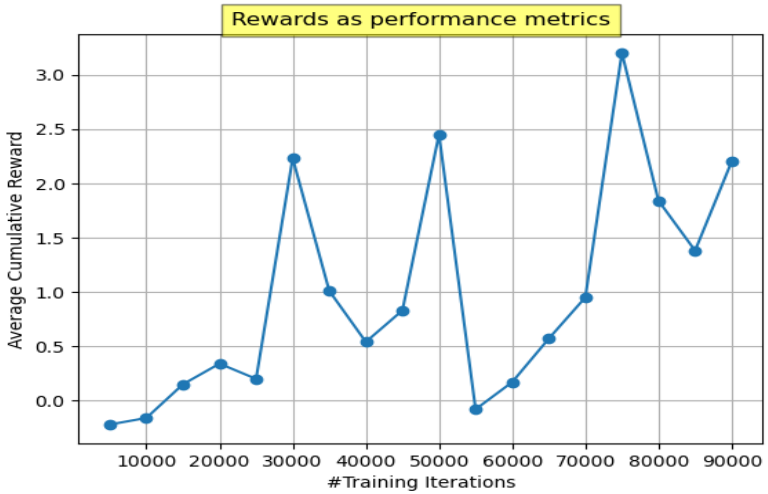
Model Parameters:

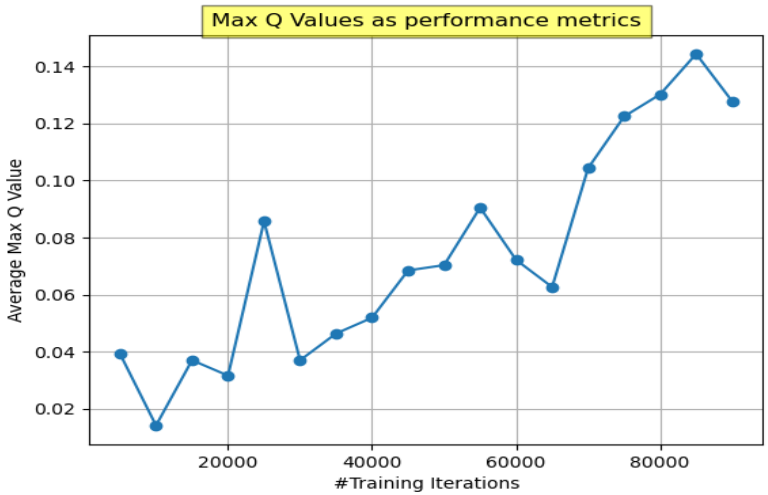
- $\gamma = 0.99$
- ϵ scheduled as in the plot below.
- $\alpha = 0.0001$





Model 2



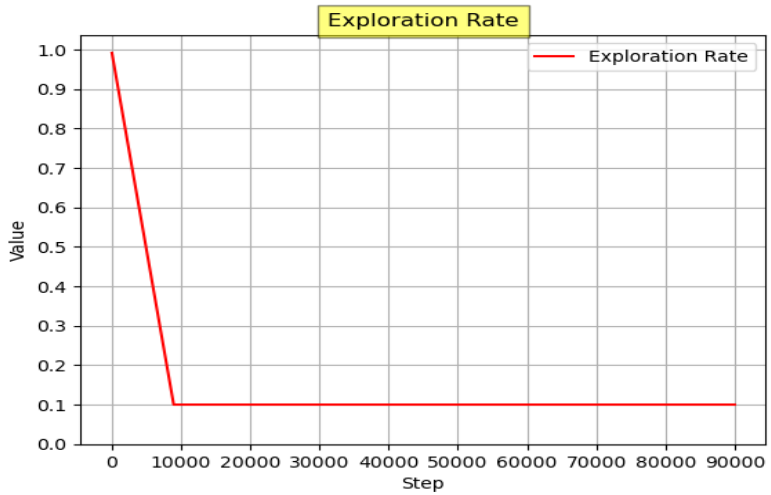


Rewards structure:

- $+0.05$ for surviving.
- -1 for "game over".

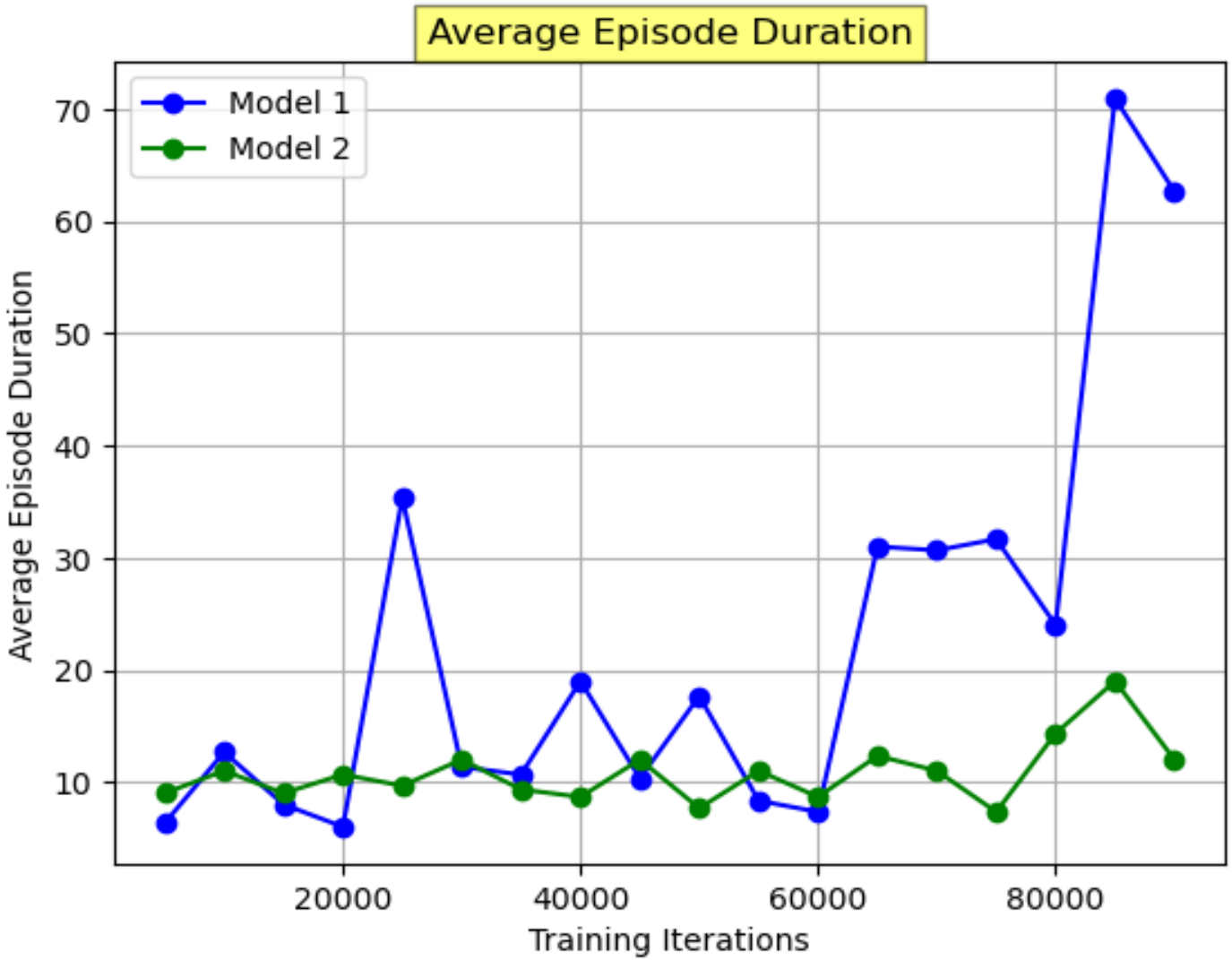
Model Parameters:

- $\gamma = 0.99$
- ϵ scheduled as in the plot below.
- $\alpha = 0.0001$



Performance Comparison (Average Episode Duration)

Each average computed over 5 episodes:



Issues and Possible Improvements

- Exploration rate is quite high. As learning progresses, it would be beneficial to fully focus on exploitation allowing the agent to reach and learn from states that are only observable in longer episodes.
- Experiment with higher learning rates for Model 1.
- Delay between consecutive frames forming a state too high, incompatible with later parts of the game where velocity consistently increases.