Jan 30, 2026

CentraleSupélec

Reinforcement Learning Project

Double Deep Q-Network for Tetris

Github Repo:
https://github.com/colinminini/RL_Tetris_Project

Team 4: Colin Minini, Auguste Malgrain, Guillaume Leteutre, Jérémy Mussote

DDQN (After-States)

Double Deep Q-Network for Tetris (Feature-Based After-State RL)

- Environment: tetris-gymnasium
- Decision level: macro-actions (place current piece)
- Representation: Dellacherie features (4D)
- Model: MLP value network $V_\theta(\phi)$
- Training: Double DQN target + replay buffer + target network

MDP Formalization

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$$

- States S: board configurations $|\mathcal{S}| \gtrsim 2^{200} \approx 1.6 \times 10^{60}$
- Actions A: primitive actions; 0 left, 1 right, 2 down, 3 rotate CW, 4 rotate CCW, 5 hard drop, 6 swap, 7 noop
- P: The world dynamics are stochastic because of the next tetromino piece
- R: Environment reward is :
  $r_t^{env} = 0.001 + 1.0 \cdot \text{lines\_cleared}_t - 2.0 \cdot \mathbf{1}[\text{game\_over}_t]$

Return, Value, and Bellman Optimality

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_t \mid S_t = s, A_t = a\right]$$

$$Q^{*}(s, a) = \mathbb{E}\left[R_t + \gamma \max_{a'} Q^{*}(s_{t+1}, a') \mid S_t = s, A_t = a\right]$$

Macro-Actions in Tetris

- Primitive control (left/right/rotate/down) yields long action sequences per piece. Instead, we define a macro-action = “place current piece”.

$$\text{Macro-action: } a = \left[\underbrace{3, \ldots, 3}_{r \in \{0,1,2,3\}}, \underbrace{d, \ldots, d}_{k \in [0,10]}, 5 \right], \quad d \in \{0, 1\}$$

- This enumerates a candidate set of placements and turns Tetris into a placement-level decision process.

After-States (Core Modeling Choice)

- Let s be the current env state (before placement). A macro-action a deterministically produces a resulting board (after placement):

$$s^{after} = f(s, a)$$

- Each candidate macro-action is simulated on a deep-copied env (simulate_sequence(copy.deepcopy(env), seq))

- We score each candidate using a shaped immediate reward plus a learned after-state value

$$V^{\pi}(s^{after}) = \mathbb{E}_{\pi}\left[G_{t+1} \mid S_{t+1} = s^{after}\right]$$

Feature Representation $\phi(s^{after})$

- Instead of pixels + CNN, we use a 4D feature vector (Dellacherie-style):

$$\phi(s^{after}) = \begin{bmatrix} \text{holes} \\ \text{bumpiness} \\ \text{aggregate\_height} \\ \text{lines\_cleared} \end{bmatrix} \in \mathbb{R}^4$$

Reward Shaping Used in Training

- Training uses shaped reward per macro-action (per placed piece), computed from feature deltas.

$$\text{line\_reward} = \begin{cases} 1 & \text{if } \ell = 1 \\ 3 & \text{if } \ell = 2 \\ 6 & \text{if } \ell = 3 \\ 12 & \text{if } \ell = 4 \\ 0 & \text{if } \ell = 0 \end{cases}$$

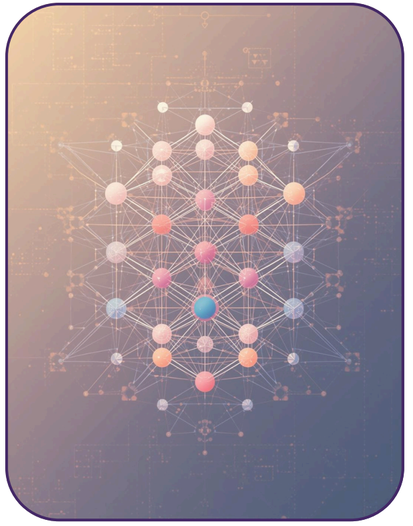
$$\Delta\text{holes} = \max\left(0, \text{holes}_t - \text{holes}_{t-1}\right)$$

$$\Delta\text{bumpiness} = \max\left(0, \text{bump}_t - \text{bump}_{t-1}\right)$$

$$r_t^{shape} = \text{line\_reward} - 2.0\,\Delta\text{holes} - 0.5\,\Delta\text{bumpiness} + 0.1$$

- only new holes and increased bumpiness are penalized.

Value Network Architecture $V_\theta(\phi)$



- The network outputs a scalar value per after-state feature vector:

$$V_\theta : \mathbb{R}^4 \to \mathbb{R}$$

- Architecture is a MLP : $4 \to 128$ (Linear) $\to$ RELU $\to 64$ (Linear) $\to$ RELU $\to 1$ (Linear)

9k parameters

$$V_\theta(\phi) = W_3\sigma\left(W_2\sigma\left(W_1\phi + b_1\right) + b_2\right) + b_3$$

$$\sigma(x) = \max(0, x)$$

Decision Rule During Training

- At each decision point, we enumerate candidates and compute shaped rewards

$$\Phi_t = \left\{ \phi_t^{(1)}, \phi_t^{(2)}, \ldots, \phi_t^{(N_t)} \right\} \quad / \quad \text{score}_t^{(i)} = r_t^{shape,(i)} + \gamma V_\theta \left(\phi_t^{(i)} \right)$$

- Training policy is ϵ-greedy over this score

$$i_t = \begin{cases} \text{Uniform}\{1, \ldots, N_t\} & \text{with probability } \epsilon_t \\ i_t^* & \text{with probability } 1 - \epsilon_t \end{cases}$$

$$i_t^* = \arg \max_{1 \le i \le N_t} \left(r_t^{shape,(i)} + \gamma V_\theta \left(\phi_t^{(i)} \right) \right)$$

- Epsilon decay formula: $\epsilon_t = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \exp\left(-\frac{t}{\text{decay}} \right)$

Replay Buffer Content

Replay buffer stores tuples:

$$(\phi_t, r_t^{shape}, d_t, \Phi_{t+1})$$

Φ_{t+1} : Feature Matrix of all candidates after-states

done_t: the terminal flag

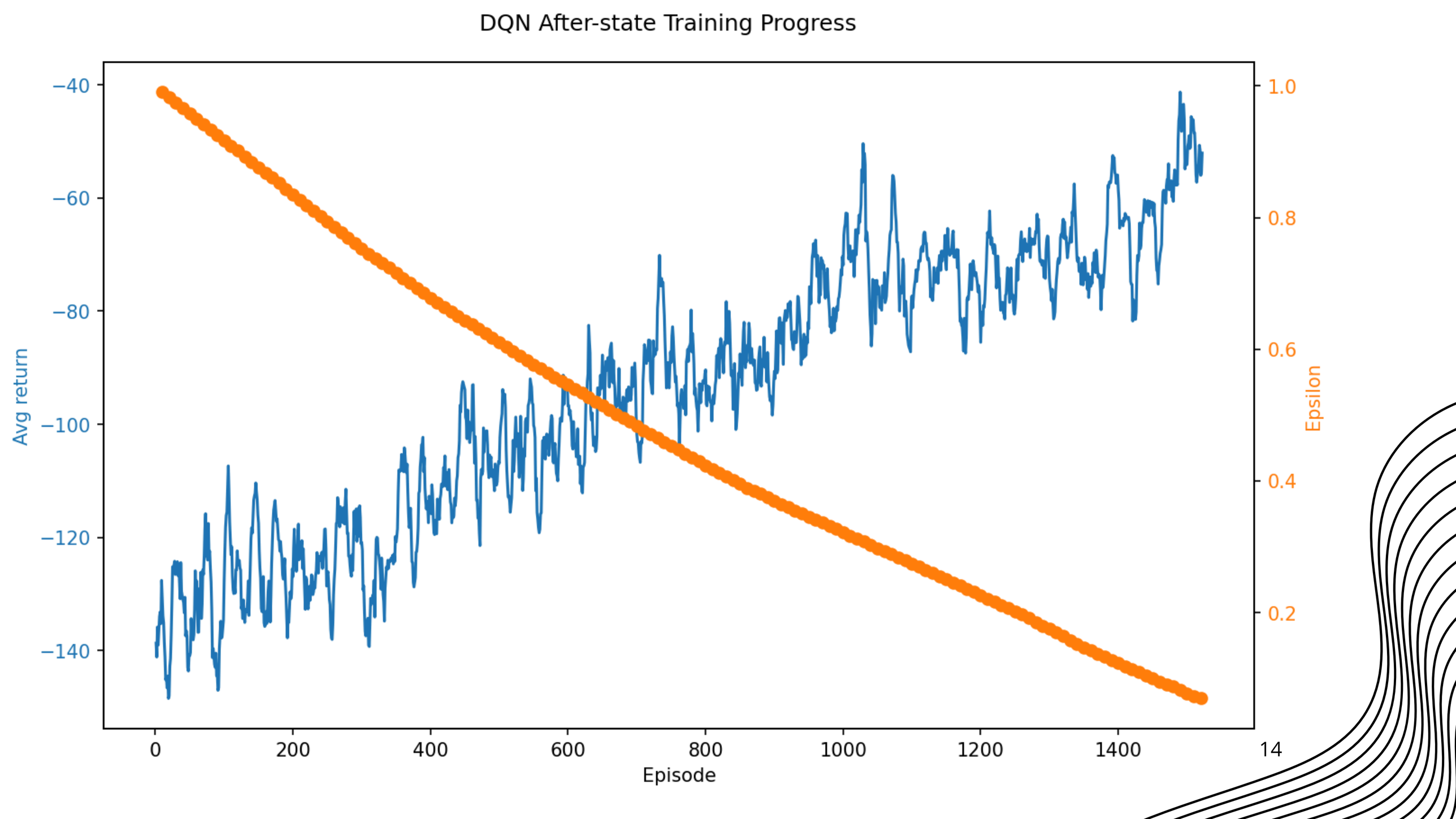
Double DQN Target

- Target definition: $$y_t = \begin{cases} r_t^{shape} & \text{if } done_t = 1 \\ r_t^{shape} + \gamma V_{\theta^-}\left(\phi_{t+1}^{(i^*_{t+1})}\right) & \text{otherwise} \end{cases}$$
- Loss function: $$\mathcal{L}(\theta) = \mathbb{E}\left[\left(V_\theta(\phi_t) - y_t\right)^2\right]$$
- Batch approximation: $$\mathcal{L}(\theta) = \frac{1}{B}\sum_{b=1}^{B}\left(V_\theta(\phi_t^{(b)}) - y_t^{(b)}\right)^2$$
- Target update periodically: $$\theta^- \leftarrow \theta$$

Optimization Details

Training settings and hyperparameters:

- Optimizer: Adam, learning rate 1×10**(-3)
- Total steps = 50k
- Batch size: 256
- Replay buffer capacity: 20k
- Start training after: Warm up until ~5 batches of transitions
- Target sync: every 500 macro-steps (Refresh target network about every ~1% of total training steps.
- Discount: γ=0.99
- eps_end = 0.02 (Good for deterministic environment like Tetris)


DQN After-state Training Progress
Avg return
−40
−60
−80
−100
−120
−140
Epsilon
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000
1200
1400
Episode

Evaluation

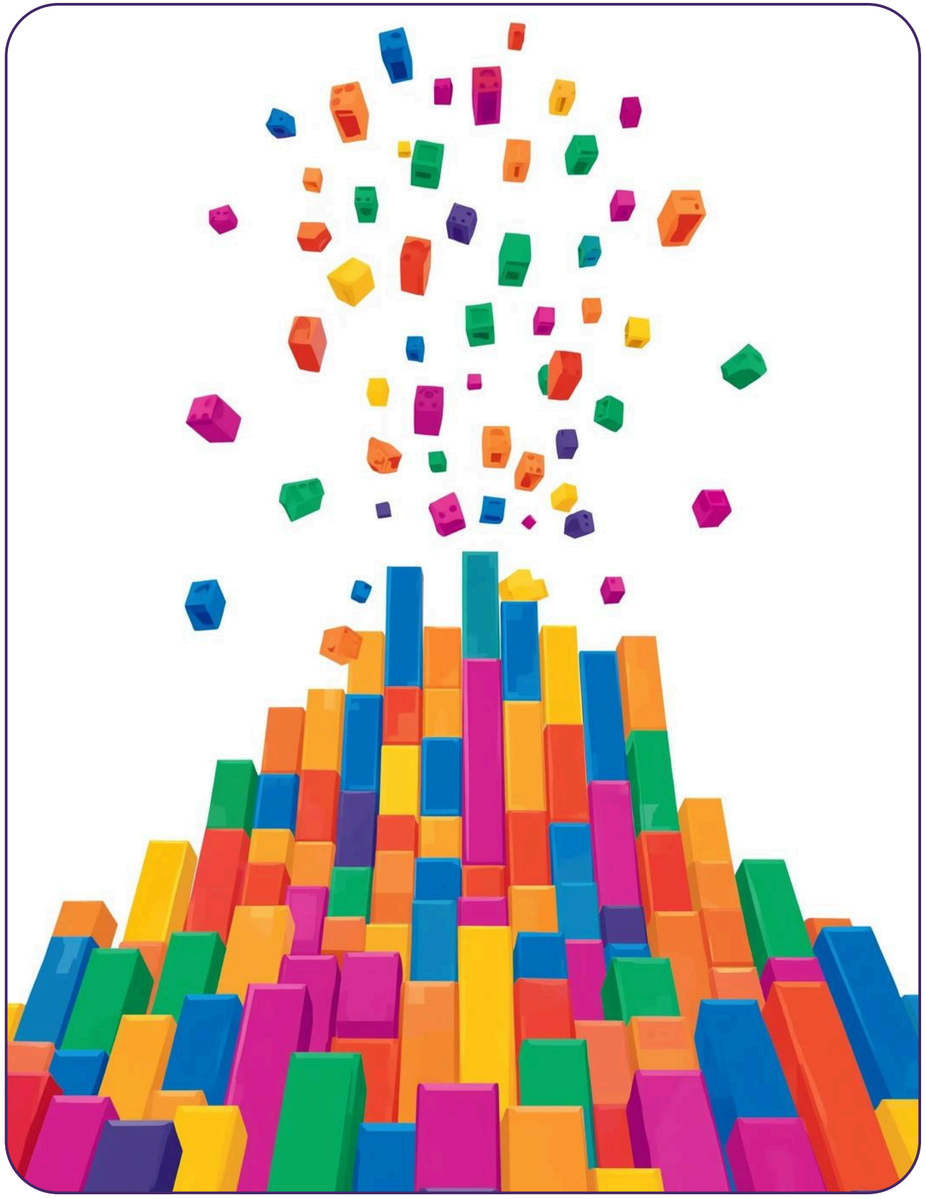
- Evaluation is greedy only and uses the Environment Return as the scoring metric

- Resulting Score: 41k (+- 17k) average Environment Return over 10 episodes

Gameplay Demo

Trained agent behavior:

- Prefers placements that avoid creating new holes
- Penalizes increases in bumpiness
- Strong line-clear preference due to shaped reward mapping



Sources

- “Playing Atari with Deep Reinforcement Learning” - Google DeepMind, 2013 - Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller
- M. van Hasselt et al., “Deep Reinforcement Learning with Double Q-learning” AAAI 2016 — Introduces Double DQN, addresses Q-value overestimation .
- G. Thiéry and B. Scherrer, “Improvements on Handcrafted Tetris Features” 2009
- Ashry (Thesis) 2020: “Applying DQN to Tetris with High-Level States”