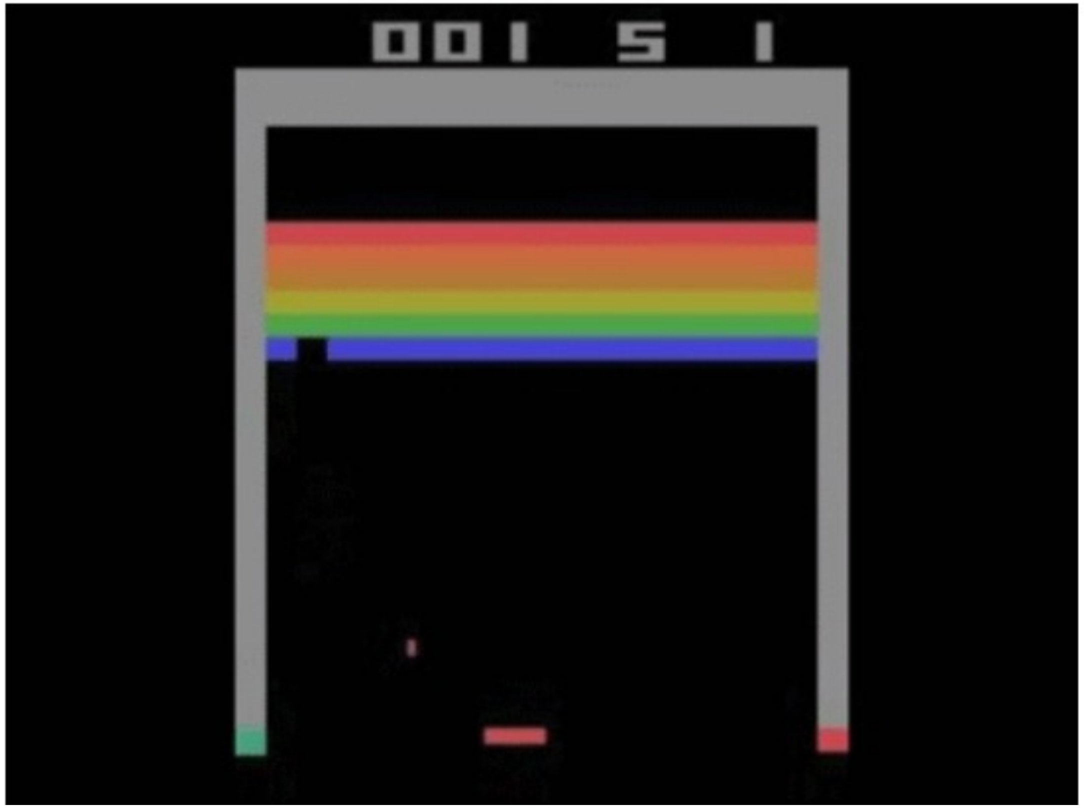
Comparative Study of Deep Q Learning and Double Deep-Q Learning on Atari Breakout

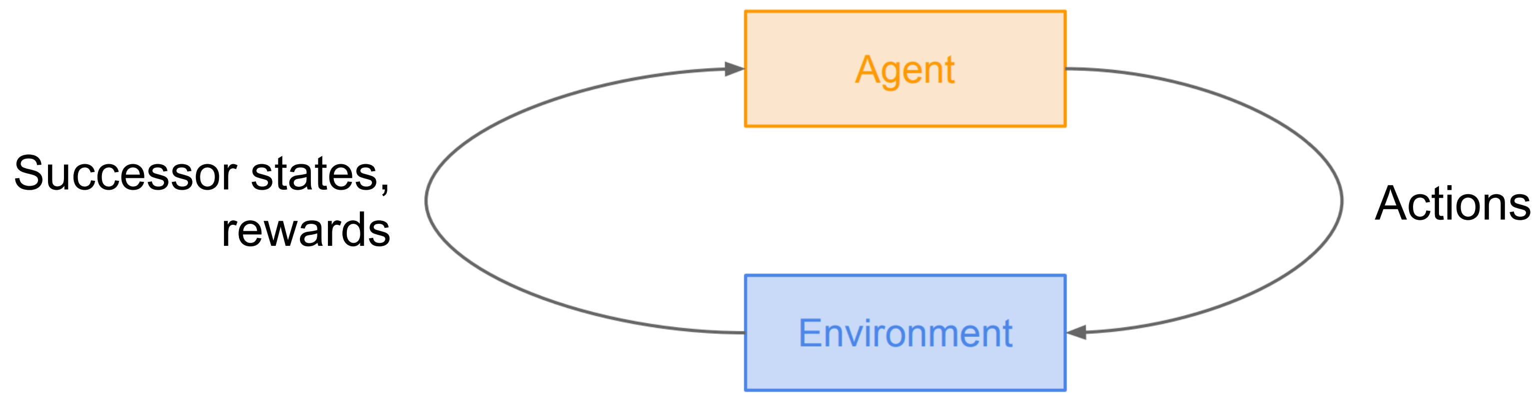


Outline

- Introduction to Reinforcement learning
- Value Function
- The Bellman equation
- Q-learning
- Deep Q networks (DQN)
- Double DQN
- Results/Discussion
- Implementation Code (If time permits)

Reinforcement learning (RL)

- Setting: agent that can take *actions* affecting the *state* of the environment and observe occasional *rewards* that depend on the state
- Goal: learn a *policy* (mapping from states to actions) to maximize expected reward over time

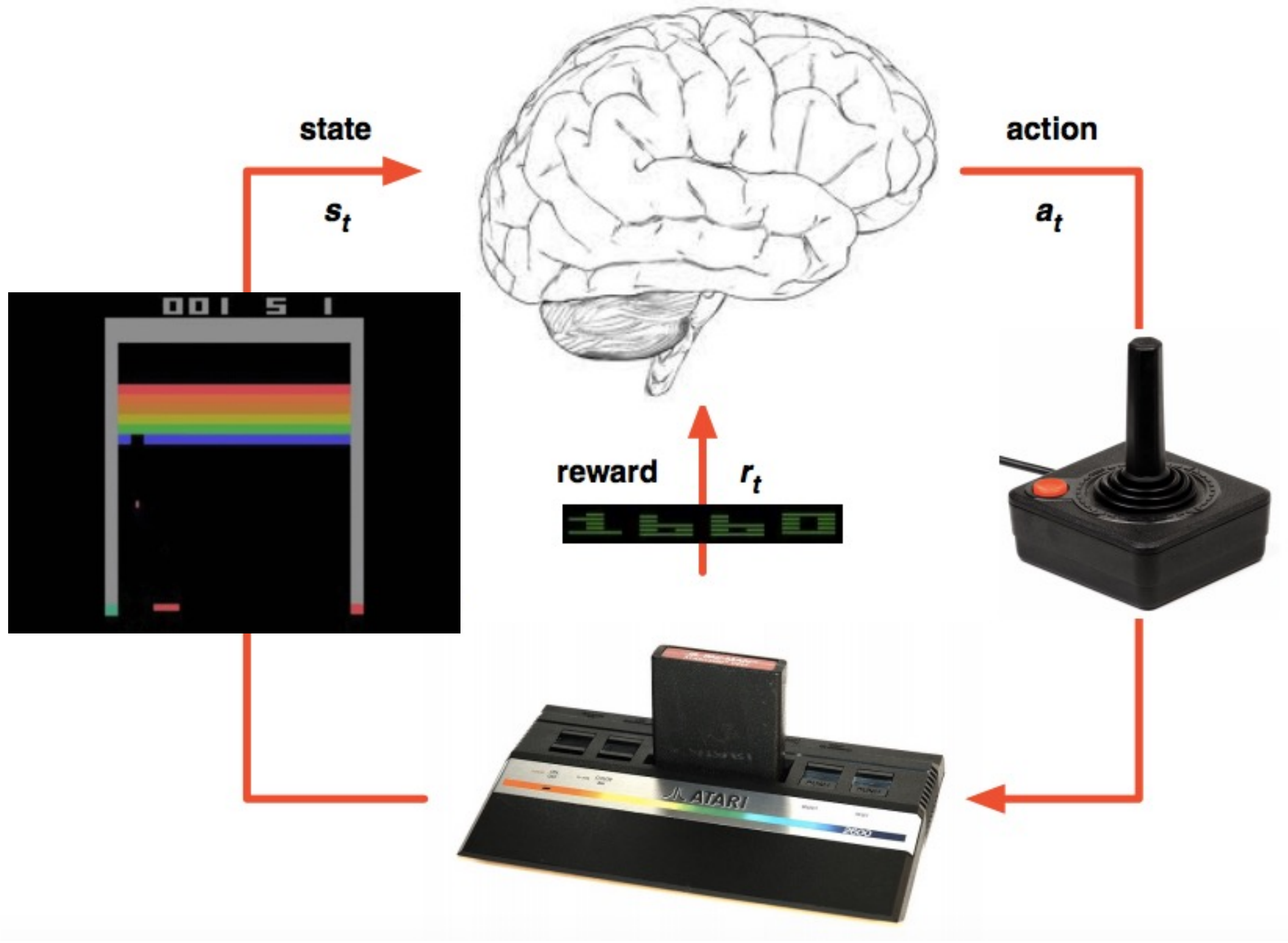


Reinforcement learning loop

- **Components**:
  - **States** s, beginning with initial state s_0
  - **Actions** a
  - **Transition model** $P(s' \mid s, a)$

    Assumption: The probability of going to s' from s depends only on s and a and not on any other past actions or states
  - **Reward function** $r(s)$
  - **Policy** $\pi(s)$: the action that an agent takes in any given state
- **Loop:**
  - From state s, take action a determined by *policy* $\pi(s)$
  - Environment selects next state s' based on *transition model* $P(s'|s, a)$
  - Observe s' and reward $r(s')$, update policy

Reinforcement learning for Atari Games



Open Gym Environment

State:

Frame as pixel values

Actions:

left ,right, fire (to reset the game when a life is lost

Reward:

Breaking Bricks (points depend on the colour of the brick)

Value function

- The *value function* $V^{\pi}(s)$ of a state s w.r.t. policy π is the expected cumulative reward of following that policy starting in s:

$$V^{\pi}(s) = \mathbb{E}_{\tau}[r(\tau) | \, s_0 = s, \pi]$$

  where τ is a trajectory with starting state s, actions given by π, and successor states drawn according to transition model: $s_{t+1} \sim P(\cdot \, | s_t, a_t)$

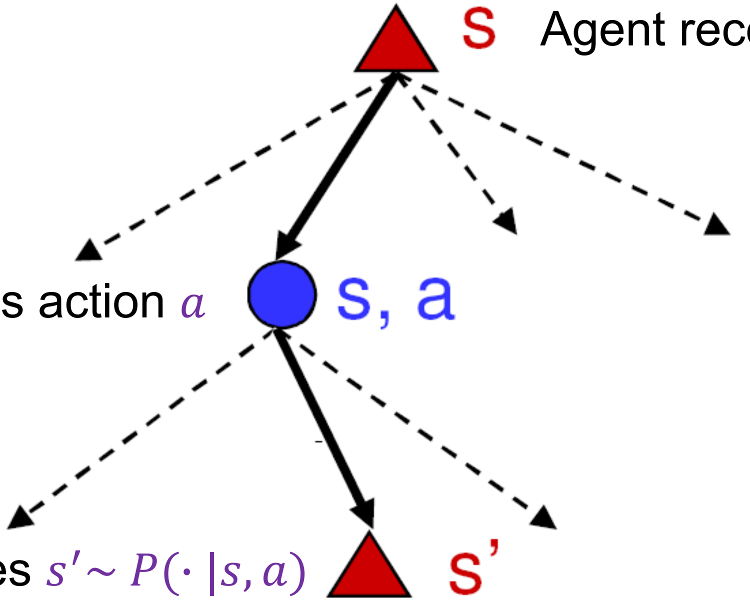
- The *optimal value* of a state is the value achievable by following the best possible policy:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

The Bellman equation

- Recursive relationship between optimal values of successive states:

$$V^*(s) = r(s) + \gamma \max_a \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s')$$



S Agent receives reward $r(s)$

Agent chooses action a

Environment chooses $s' \sim P(\cdot\,|s,a)$

Expected reward: $V^*(s')$

Optimal policy:
$\pi^*(s) = \arg\max_a \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s')$

Discounting

- **Problem:** the sum of rewards of individual states is not normalized w.r.t. sequence length and can even be infinite
- **Solution:** define cumulative reward as sum of rewards *discounted* by a factor γ, $0 < \gamma \leq 1$

Discounting

- *Discounted cumulative reward* of trajectory $\tau = (s_0, s_1, s_2, s_3, \dots)$:

$$r(s_0, s_1, s_2, s_3, \dots) = r(s_0) + \gamma\, r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + \cdots$$
$$= \sum_{t \geq 0} \gamma^t\, r(s_t)$$

- Sum is bounded by $\frac{r_{\max}}{1-\gamma}$ (assuming $0 < \gamma \leq 1$)
- Helps algorithms converge
- Notice:

$$\underbrace{r(s_0, s_1, s_2, s_3, \dots)}_{\text{Cumulative reward of trajectory starting at } s_0} = \underbrace{r(s_0)}_{\text{Reward at } s_0} + \gamma\, \underbrace{r(s_1, s_2, s_3, \dots)}_{\text{Discounted reward of trajectory starting at } s_1}$$

Q-Learning

- Relationship between regular values and Q-values:

$$V^*(s) = \max_a Q^*(s,a)$$

- Regular Bellman equation:

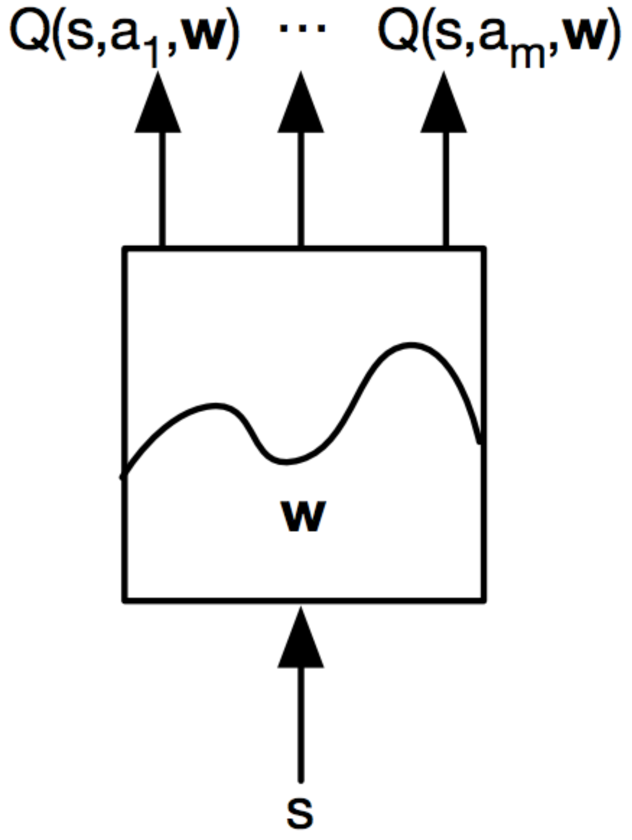
$$V^*(s) = r(s) + \gamma \max_a \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s')$$

- Bellman equation for Q-values:

$$Q^*(s,a) = r(s) + \gamma\, \mathbb{E}_{s' \sim P(\cdot|s,a)} \max_{a'} Q^*(s',a')$$
$$= \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r(s) + \gamma \max_{a'} Q^*(s',a') \right]$$

Deep Q-learning

- Train a deep neural network to estimate Q-values:



Deep Q-learning

$$Q^*(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

- At each step of training, update model parameters w to "nudge" the left-hand side toward the right-hand "target":

$$y_{\text{target}}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q_{\text{target}}(s',a')|s,a\right]$$

- Loss function:

$$L(w) = \mathbb{E}_{s,a}\left[\left(y_{\text{target}}(s,a) - Q_w(s,a)\right)^2\right]$$

Deep Q-learning

- Target: $y_{\text{target}}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[r(s) + \gamma \max_{a'} Q_{\text{target}}(s',a')|s,a\right]$
- Loss: $L(w) = \mathbb{E}_{s,a\sim\rho}\left[\left(y_{\text{target}}(s,a) - Q_w(s,a)\right)^2\right]$

- Gradient update:

$$\nabla_w L(w) = \mathbb{E}_{s,a\sim\rho}\left[\left(y_{\text{target}}(s,a) - Q_w(s,a)\right)\nabla_w Q_w(s,a)\right]$$
$$= \mathbb{E}_{s,a\sim\rho,s'}\left[\left(r(s) + \gamma \max_{a'} Q_{\text{target}}(s',a') - Q_w(s,a)\right)\nabla_w Q_w(s,a)\right]$$

- SGD training: replace expectation by sampling transitions (s,a,s') using *behavior distribution* and *experience replay*

Deep Q-learning in practice

- Training is prone to instability
  - Unlike in supervised learning, the targets themselves are moving!
  - Successive experiences are correlated and dependent on the policy
  - Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution
- Solutions
  - Use *experience replay*
  - Freeze target Q network (Double-DQN)

Deep Q-learning in practice – Final Implementation

- At each time step:
  - Take action a_t according to *epsilon-greedy policy*
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*

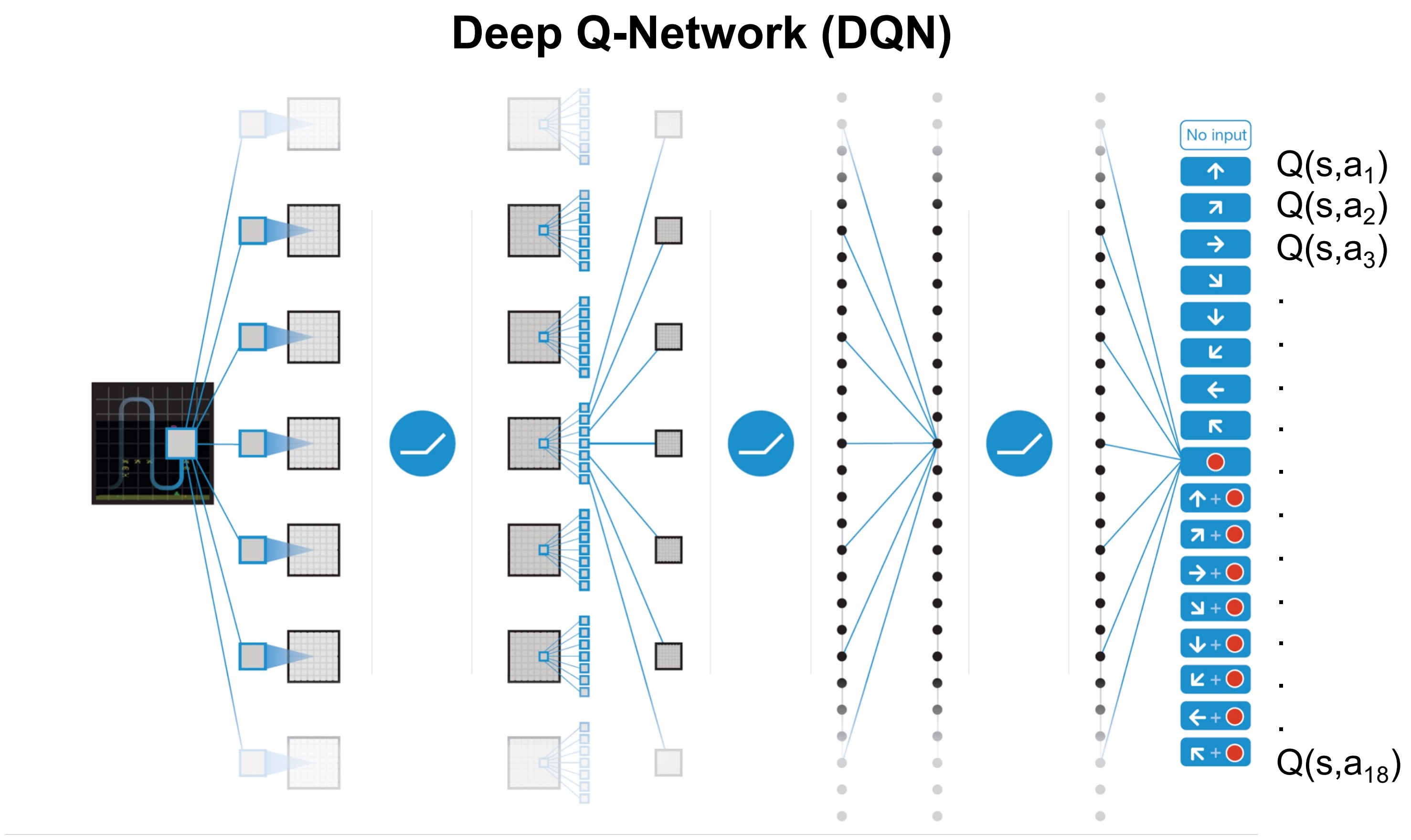
s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\cdots$
$s_t, a_t, r_{t+1}, s_{t+1}$

  - Randomly sample *mini-batch* of experiences from the buffer
  - Perform gradient descent step on loss:

$$L(w) = \mathbb{E}_{s,a,s'}\left[\left(r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q_w(s,a)\right)^2\right]$$

Deep Q-learning in Atari

- End-to-end learning of $Q(s,a)$ from pixels s
- Output is $Q(s,a)$ for 18 joystick/button configurations
- Reward is change in score for that step



Double Deep Q-learning

- Max operator in standard Q-learning is used both to select and evaluate an action, leading to systematic over-estimation of Q-values
- Modification: *select* action using the online (current) network, but *evaluate* Q-value using the target network
- Regular DQN target:

$$y_{\text{target}}(s,a) = r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a')$$

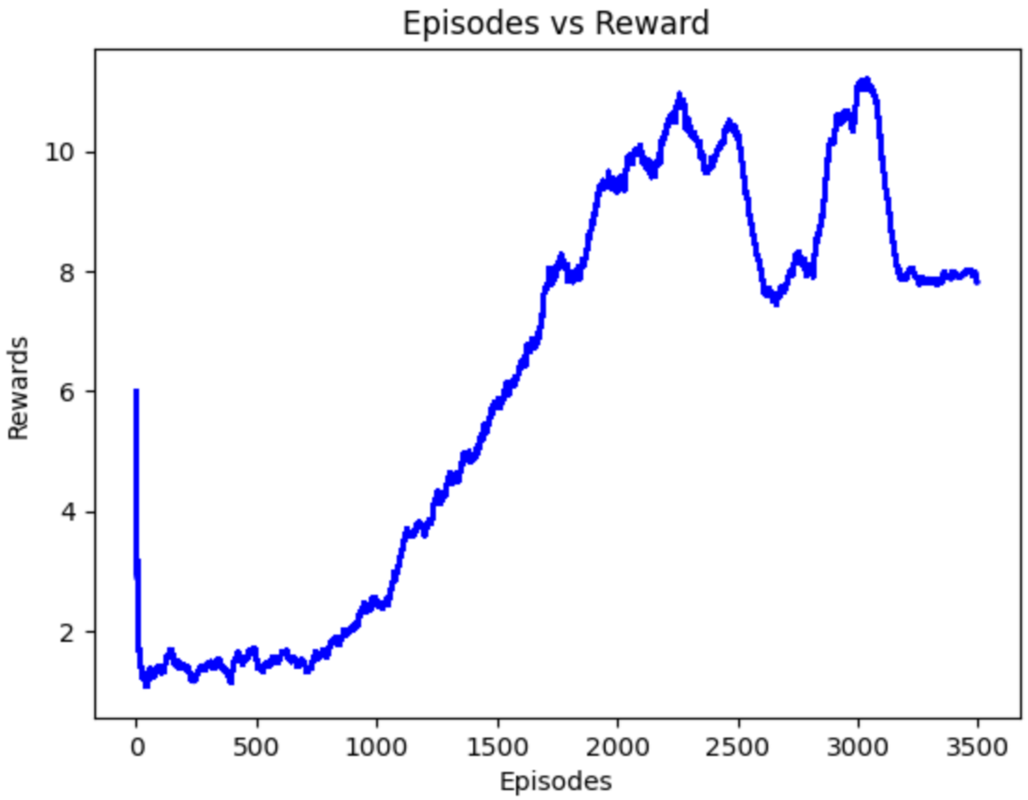
- Double DQN target:

$$y_{\text{target}}(s,a) = r(s) + \gamma\, Q_{\text{target}}\left(s', \operatorname{argmax}_{a'} Q_w(s', a')\right)$$

Results

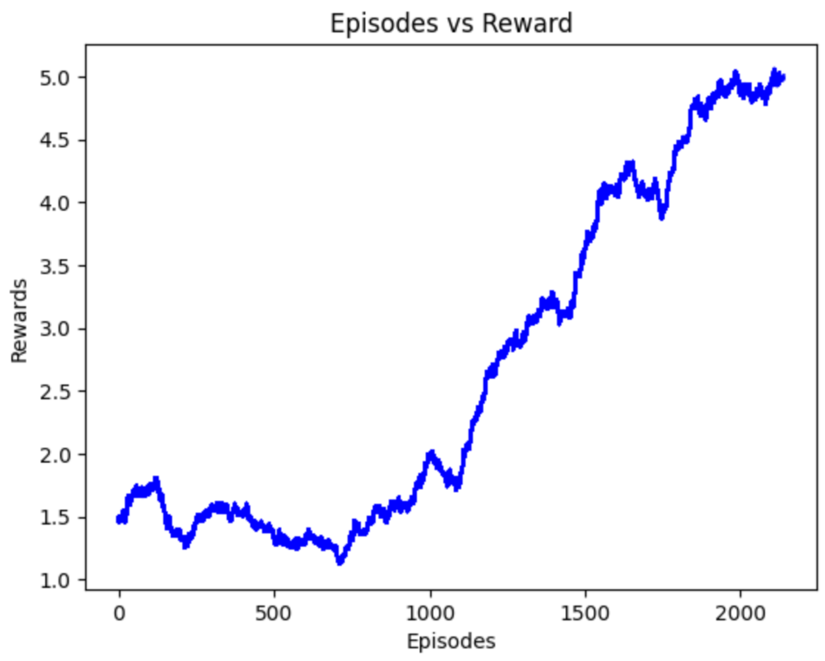
DQN Results

Double - DQN Results

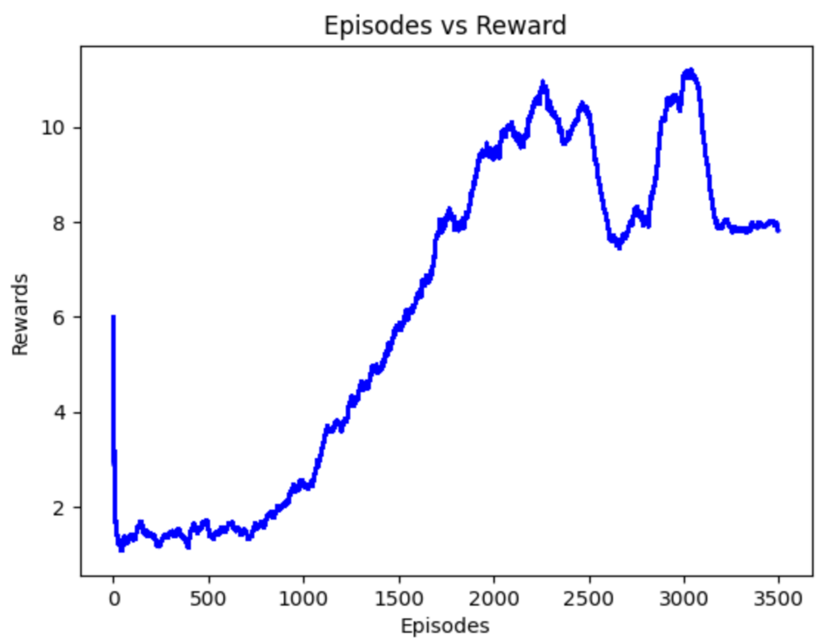


Discussion – Learning Trends

DQN Results



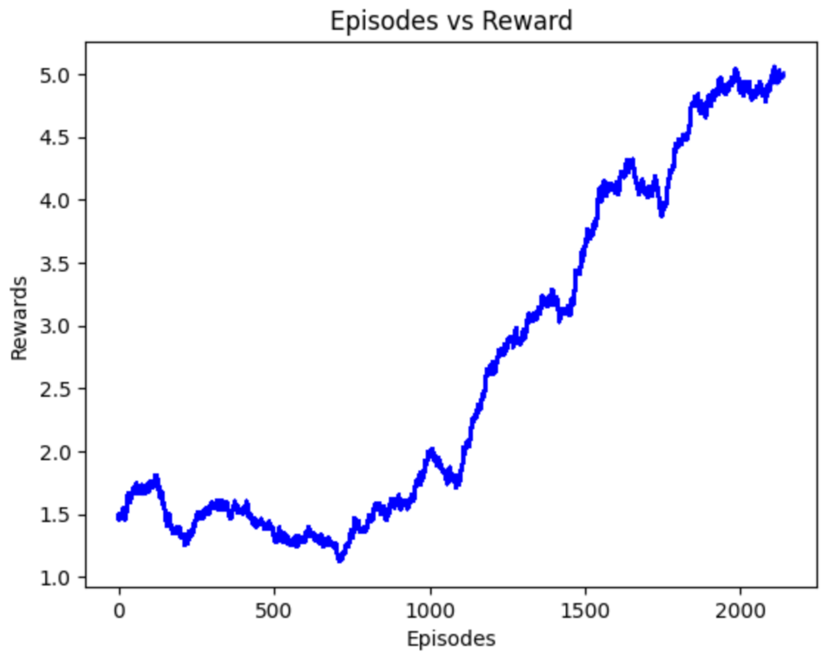
Double - DQN Results



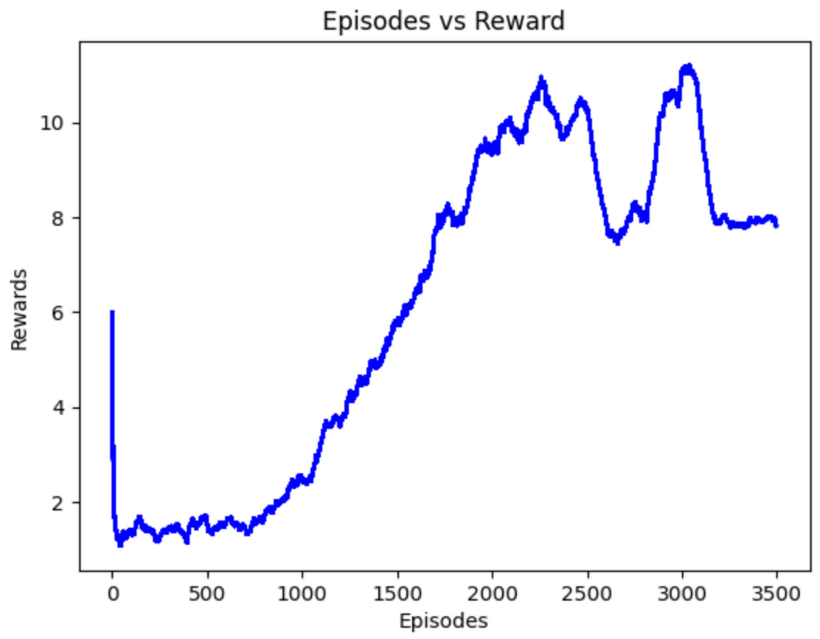
Both DQN and DDQN show an increasing trend in rewards over episodes, but DDQN achieves a higher peak reward than DQN i.e., it performs better by mitigating the overestimation bias present in DQN

Discussion – Volatility in Rewards

DQN Results



Double - DQN Results



DDQN has more volatility in the reward values compared to the DQN (in the end)
This could indicate that DDQN is exploring more diverse strategies / over-exploration , overfitting to particular states, or is more sensitive to the randomness in the environment which indicates need for more fine tuning of params

Visualization of Learned Policy

Link to the Project Code and my review paper

Code: https://github.com/vporwal3/Deep-Reinforcement-Learning/tree/main

Review Paper: https://github.com/vporwal3/Deep-Reinforcement-Learning/blob/main/RLpaper.pdf